

For Reference

NOT TO BE TAKEN FROM THIS ROOM

EX LIBRIS
UNIVERSITATIS
ALBERTAENSIS





Digitized by the Internet Archive
in 2020 with funding from
University of Alberta Libraries

<https://archive.org/details/McCalla1970>

THE UNIVERSITY OF ALBERTA

WISE: A MODEL TO UNDERSTAND SIMPLE ENGLISH

by



Gordon Irvine McCalla

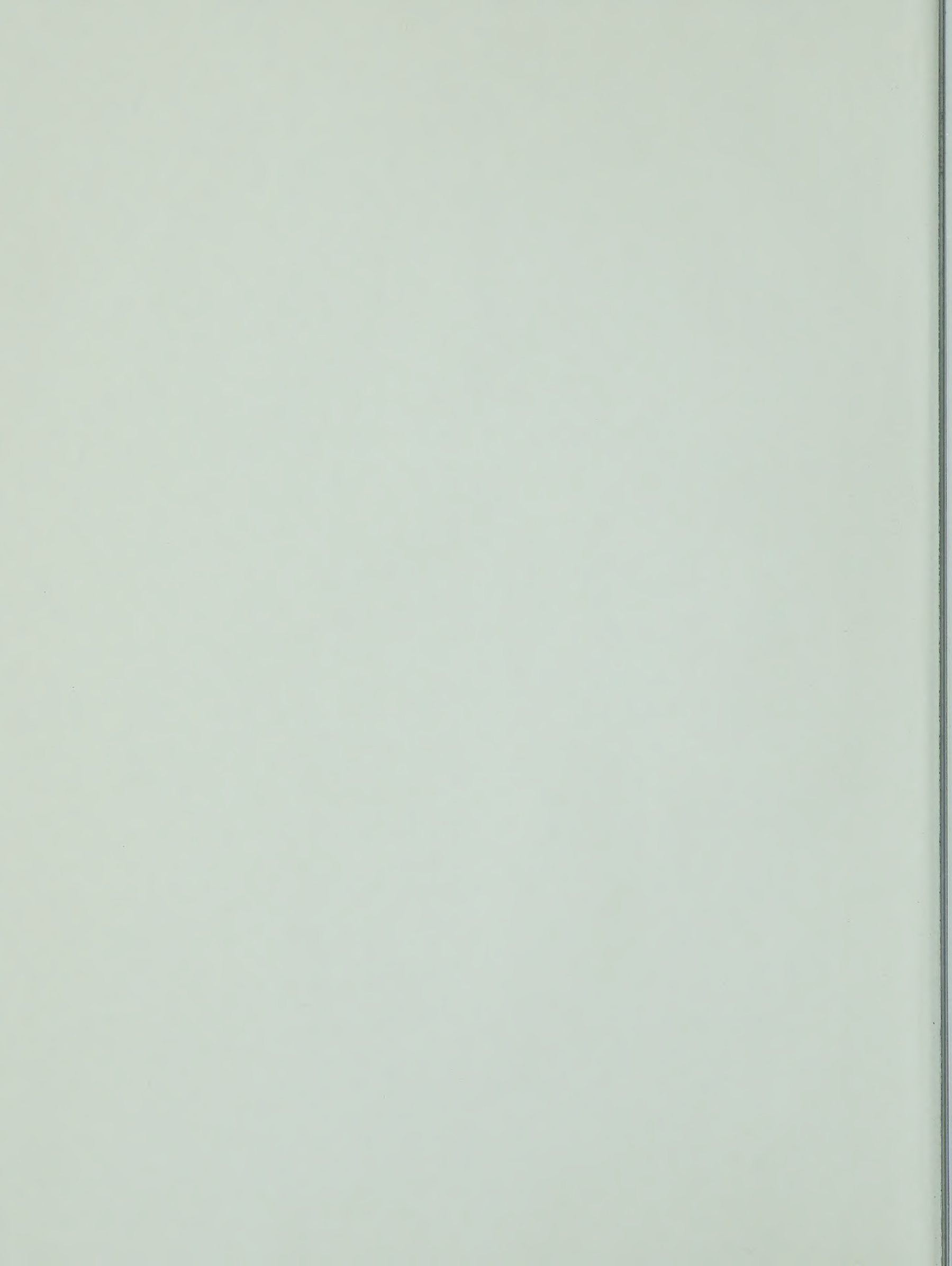
A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1978



THE UNIVERSITY OF ALBERTA

MUSE: A MODEL TO UNDERSTAND SIMPLE ENGLISH

by



Gordon Irvine McCalla

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE
OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1970

THE UNIVERSITY OF ALABAMA

DEPARTMENT OF GEOGRAPHY

BY

ROBERT L. HARRIS



A THESIS

PRESENTED TO THE FACULTY OF THE UNIVERSITY OF ALABAMA

IN CANDIDACY FOR THE DEGREE OF MASTER OF ARTS

OF THE UNIVERSITY OF ALABAMA

UNIVERSITY OF ALABAMA LIBRARY

LIBRARY OF THE UNIVERSITY OF ALABAMA

1968

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read,
and recommend to the Faculty of Graduate Studies for
acceptance, a thesis entitled MUSE: A MODEL TO
UNDERSTAND SIMPLE ENGLISH, submitted by Gordon Irvine
McCalla in partial fulfilment of the requirements for
the degree of Master of Science.

Date .. *July 21, 1970*

ABSTRACT

This thesis is concerned with computer modelling of natural language. Some recent natural language models are reviewed; and a particular model, Quillian's (1969) Teachable Language Comprehender, is examined in detail. The thesis then proposes MUSE, a natural language processor that uses both syntactic and semantic techniques to understand simple English input. MUSE can act upon its understanding of the input and can use this understanding to increase its "knowledge of the world" (stored in a highly interconnected memory network). MUSE's theory and implementation are thoroughly discussed, and an evaluation of the model is given.

ACKNOWLEDGEMENTS

I wish to express my gratitude to my supervisor, Dr. J.R. Sampson, who has provided valuable assistance at every stage in the development of this project. I am indebted, also, to the National Research Council of Canada for its financial support, and to Dr. D.B. Scott, Chairman of the Department of Computing Science, University of Alberta, who has furnished both funds and computer facilities. Finally, I would like to thank Mrs. Mary Yiu for typing the final draft of the thesis.

TABLE OF CONTENTS

	Page
CHAPTER I NATURAL LANGUAGE MODELLING	1
CHAPTER II THE TEACHABLE LANGUAGE COMPREHENDER	23
CHAPTER III FORMULATION OF MUSE	39
CHAPTER IV IMPLEMENTATION OF MUSE	79
CHAPTER V EVALUATION OF MUSE	102
REFERENCES	113
APPENDIX I MEMORY FILES	118
APPENDIX II SAMPLE RUN	120

LIST OF FIGURES

Figures		Page
2.1	The memory concept for RUFF1	25
2.2	The memory concept for RUFF4	34
2.3	Output from TLC	34
3.1	Flow diagram for MUSE	44
3.2	A parse tree	52
3.3	Parse tree filled with candidate combination	53
3.4	A-action of the third type	55
4.1	Parse tree in SYNTAX notation	83
4.2	Parse tree in English	84

CHAPTER I

NATURAL LANGUAGE MODELLING

Introduction

This thesis is about the study of language using simplified linguistic models. Major emphasis is placed on computer models, especially those which deal heavily with semantic processing. Eventually an original natural language computer model is proposed which understands simplified written English input and acts upon it in an appropriate manner. It is called MUSE, a Model to Understand Simple English. Before MUSE can be fully understood, however, a closer examination of natural language modelling is needed.

Man has been curious about language ever since he first began to communicate. Scholars have studied the structure of language from at least the time of the Greeks, when the first understanding of the role of parts of speech began to emerge. Knowledge of language has developed steadily since then, culminating in the sophisticated memory networks and transformational grammars of today; but much more still has to be learned before any language can be fully understood.

Not only is the study of language important in its own right, but it also gives great insight into many closely related disciplines, prime among them psychology. When using language, people translate it into some sort of cognitive structure. Thus, a knowledge of language should give the psychologist or psychiatrist some insights into the processes taking place behind the words. A thorough knowledge of language would also enable many useful things to be implemented, among them mechanical translation between different languages, natural language programming of computers, context sensitive information retrieval systems, and meaningful question answering and problem solving machines.

Natural language is generally considered to consist of three components: syntax, semantics, and phonology. The syntactic component is concerned with the "principles for combining words to form grammatical sentences" (Langacker, 1967, page 10); that is, how words are related to each other in a sentence. Syntactic analysis of a sentence often calls for the sentence to be "parsed" into a tree structure representing the connections of words in the sentence to each other. This is usually accomplished by using transformation rules on the parts of speech of the words in the sentence.

Semantics is the study of meaning, that is how words are related to the speaker's "knowledge of the world". Thus, a semantic analysis of a sentence would involve associating the words in the sentence with their meanings.

Finally, the phonological component of natural language deals with phonetics, the sound sequences emitted when speaking words in a sentence. This thesis will not be concerned with the phonological part of language, but solely concerned with written English (from now on the terms "natural language" and "language" will usually refer to written English).

Of course, the three component structure is grossly oversimplified. The border between all three groups is not well defined, and even within each group there are many as yet unresolved difficulties. Among the most troublesome problems are ambiguities. These can be syntactic, where there is more than one possible grammatical analysis for the sentence; or semantic, where there are several valid interpretations of the meanings of the words in the sentence. A good deal of research in modern linguistics is concerned with explaining ambiguity.

Because of the complexity of language, simplifying assumptions are often required in attempts to model natural language. Many natural language models have been developed to consider aspects of syntax and semantics, and these are often tested by being implemented as computer models. Computer modelling enables a preciseness of thought seldom before achievable. The development of symbol oriented list processing and string manipulation languages (such as LISP or SNOBOL) for the computer has hastened even further the swing towards programming computer models of language.

Components of a Linguistic Model

Before a computer model can be implemented numerous problems must be examined and solved. The modeller must decide on the particular aspect of language he wants to simulate, and he must have some specific problem area to constitute the "universe of discourse" for the model. He needs to devise a memory structure in which to store the model's "knowledge of the world". He must decide the sort of restrictions to impose on the input to the model, and how this input is to be interpreted by the model. The modeller must choose what is to be produced as output from the model; and, if the system is to be of much generality, he must devise ways of increasing

and modifying the model's memory and procedures. Finally, he must choose an effective method for testing his model. Each of these problems is described below, along with some of the diverse solutions which have been devised.

General Problem Areas for the Model

The first thing the researcher must do is decide the general problem area in which he wants to build a model. There are several areas of research where natural language plays an important role, question answering and fact retrieval being perhaps the most widespread. Such systems usually consist of a highly structured data base which contains information about some limited project. Answers to questions are derived from this data base using some set of procedures. Usually it would be very convenient to be able to ask questions in natural language and get natural language responses. Simmons (1970) outlines a reasonably complete survey of recent efforts in this field.

The second major area for research is problem solving. Closely related to question answering, problem solving usually involves more emphasis on mathematical procedures and less stress on deducing answers from a data base. It would still be attractive, however, to be able to state the problem in natural language.

Bobrow (1968) and Charniak (1969) solve algebra and calculus word problems stated in relatively normal English.

Psychological modelling is another field where natural language processing plays a large role. The idea here is to simulate the cognitive processes underlying behavior with some sort of highly interconnected memory net. Programming the model to process natural language input and produce natural language output aids in discovering important principles of psychology. Colby and Smith (1969); Colby, Tesler, and Enea (1969); and Abelson and Reich (1969) have produced papers which describe the latest versions of a long line of "human belief system" models.

Finally, natural language models are built to help in the study of language itself. Models have been devised that handle many aspects of syntax and semantics. Transformational grammars have been programmed (Dewar, Bratley, and Thorne, 1969; Friedman, 1969) and semantic memory systems devised (Quillian, 1969). Combinations of these have been implemented by Schank, Tesler, and Weber (1970) and Schwarz, Burger, and Simmons (1970). Competence models of linguistics have been proposed (Katz and Postal, 1964;

Chomsky, 1965), but these are not strictly relevant to the performance models emphasized in this thesis.

Most of the above areas overlap and most models have been devised with more than one problem area in mind. For example Quillian (1969) describes a model limited mostly to developing a semantic memory net, but he calls it a "psychological theory" and outlines ideas for incorporating new features to improve his system.

Specific Subjects to be Handled by the Model

During the discussion of the general areas of research, the problem of specific subject areas has already been touched upon. Once the researcher has decided on his general interest (or interests) he usually chooses some smaller topic to form the data base for his system. Aside from the algebra and calculus word problems and human belief systems already mentioned, there are many other subjects which have been explored. Weizenbaum (1966) developed ELIZA, a conversation machine programmed to simulate a psychiatric interview; and Becker (1969) has devised a method to handle analogy and induction. Systems have been developed to solve problems about geometric relationships in pictures (Coles, 1969; Kochen 1969) and to produce output on airline flight schedules (Woods, 1968). Question answering systems respond to

queries on varying subjects including survey data on large U.S. cities (Kellogg, 1968), military supply problems (Craig, Berezner, Carney, and Longyear, 1966), kinship relations (Lindsay, 1963; Shapiro and Woodmansee, 1969), baseball schedules (B.F. Green, Wolf, Chomsky, and Laughery, 1963), and children's stories (Tharp and Krulee, 1969). In addition many natural language models handle a variety of self contained problems. Thus, Raphael's (1968) SIR can handle any problem whose basic postulates have been fed to the system.

Memory Structure of the Model

Once the researcher has defined what subject will form his model's "knowledge of the world", he must design a memory system in which to store this knowledge. Two main types of memory have been employed: the procedure oriented memory and the network oriented memory.

The procedure oriented memory systems rely on rules to carry the burden of interpretation and so are themselves relatively unstructured. Classical linguistic analysis depends on just this sort of approach. The memory consists of a lexicon (or dictionary) containing valid words and corresponding syntactic and semantic markers. A separate set of "rewrite" rules for handling these words is defined as well. Rosenbaum (1967) proposes

such a system. Woods (1968), Kellogg (1968), Kochen (1969), and Weizenbaum (1966) have designed procedure oriented models which differ from the linguistic models only because the rules for handling the lexicon are not separate from the lexicon. Instead, each word in the dictionary can have pointers to various procedures to be executed when that word is encountered.

The network oriented systems generally define a much more sophisticated memory structure, usually comprised of two parts: a dictionary and a network. The dictionary contains the natural language words (and corresponding markers of various types) which the system recognizes. The network is an interrelated structure representing concepts and relations in the system and consisting of nodes and links connecting the nodes. Pointers extend from words in the dictionary to nodes in the network.

Many early natural language processors such as BASEBALL (B.F. Green et al., 1963) and SAD-SAM (Lindsay, 1963) restrict the network to tree structures of one-way unlabelled links. This was followed by a trend towards ring structures which are two way links allowing the memory to be defined in a highly recursive manner. DEACON (Craig et al., 1966) is one such system. Recent efforts have not only allowed the links to branch

recursively back into the network, but have also labelled the links, thus enabling very complex, nested relational sentences to be properly interpreted. Some good examples of such systems are Shapiro and Woodmansee's (1969) SAMENLAQ II, PROTOSYNTHESIS III by Schwarz et al. (1970), Quillian's (1969) TLC and SPINOZA II by Schank et al. (1970). The belief system of Colby and Smith (1969) and Colby, Tesler, and Enea (1969), and Becker's (1969) model go still a step further by weighting each link in the memory according to its "importance" to the system.

Input to the Model

Defining the input for a natural language model is far from being an easy task. Not only are an unlimited number of grammatical sentences possible in any language, but the definition of grammatical itself varies according to the dialect under consideration. In addition computer modelling necessitates numerous simplifying assumptions so that space and time constraints can be satisfied. Clearly, some restrictions must be imposed on the input.

Natural language models so far devised seem to fall into several categories of restricted input. The most rigid systems are those which use a completely artificial language to avoid the vagaries of natural language. One

such is the QA3 system devised by C. Green (1969) which uses the predicate calculus as input.

Input is sometimes restricted to handling only simple sentences. Hence, SAMENLAQ II (Shapiro and Woodmansee, 1969) and Abelson and Reich's (1969) belief system both assume simple "X relation Y" sentence input. Closely related are the systems which employ a paradigmatic approach, i.e. use fill-in-the-blank sentence types where the blanks are variables to be replaced by words. Thus, "X is a Y" is a sentence form for equivalence, X and Y being the variables standing for words. Raphael's (1968) SIR, Bobrow's (1968) STUDENT, Charniak's (1969) CARPS, and Colby and Smith's (1969) ABS1 all have restrictions of this type on the input.

Another way of reducing input complications is to restrict the input to sentences on a specific topic. These systems often work on a "keyword" principle which makes a wide variety of sentences possible as long as the subject matter remains constant. Good operation of the ELIZA (Weizenbaum, 1966), BASEBALL (B.F. Green et al., 1963), and SAD-SAM (Lindsay, 1963) systems depends on the input being on a limited topic.

When a truly far reaching natural language model is desired, however, such "patchwork" methods are not sufficient, and a more general scheme is required.

Usually this involves defining a set of rules to apply to the input to reduce it to some convenient form. Traditionally these rules have been formally defined generative and transformational rewrite rules. Thus, the input to the question answering system of Tharp and Krulee (1969), and Kochen's (1969) diagram processor must be "parseable" English sentences.

Finally, there are the models which claim to have unrestricted input, in principle. These models do not tend to have rewrite rules as rigidly defined as those of transformational grammars. Whenever a new sentence does not fit the existing structures, new rules are defined to handle these cases. Of course, any working system will not be complete and will have many sentences which are not understandable to that particular version. Recent systems along these lines are SPINOZA II (Schank et al., 1970), PROTOSYNTHESIS III (Schwarz et al., 1970), and TLC (Quillian, 1969).

Interpreting the Input

The most important aspect of any natural language processor is how it interprets its input. Clearly there is a need for the system to in some sense "understand" an input sentence. This involves translating the sentence into a form which the model has been equipped to handle.

Interpreting the input breaks down into three interwoven subprocesses: syntactic processing, semantic processing, and deductive processing. It is impossible to truly separate these entities from each other, but a separate analysis does give some insights into their relationship to natural language processing and to each other.

Syntactic processing usually involves discovering which words in a sentence are related to each other, and in a limited sense how they are related. It is the job of any model's syntactic element to weed out interpretations of a sentence which are not "grammatical". The easy way to do this is to restrict input to sentences that by definition are grammatical and need little or no syntactic processing to be understood by the model. This is the case with models that limit input to an artificial language, to simple sentences, or to form sentences (see the previous section for examples).

Another common method of syntactic analysis is to use phrase structure or transformational grammars to produce trees which represent the syntactic connections possible in a sentence. Many computer programs have been written to produce syntactic trees (although most of them handle only restricted English input), and some of them are used in natural language processors devised by

B.F. Green et al. (1963), Woods (1968), Kellogg (1968), Coles (1969), Tharp and Krulee (1969), Lindsay (1963), and Kochen (1969).

Sometimes the syntax portion of a model is merged with other elements, and syntactic analysis is a by-product of the application of rules which can be both syntactic and semantic. PROTOSYNTHESIS III (Schwarz et al., 1970) and SPINOZA II (Schank et al., 1970) both exemplify what seems to be a growing trend away from separate syntactic and semantic analyses.

Semantic processing of language becomes necessary when there is more than one syntactic analysis (syntactic ambiguity), or when words in the sentence have more than one meaning (semantic or contextual ambiguity). Semantics is not nearly as well researched a field as is syntax. The study of meaning has been in confusion for years, and in the past semantic processing most often consisted only of attaching semantic markers to words to enable better syntactic processing. Today there is renewed interest in meaning, and almost all recent computer models employ some reasonably sophisticated techniques for semantic processing.

Just as syntactic processing aims to keep only grammatical interpretations of the input, semantic analysis attempts to retain only "semantically well-formed"

(Simmons, 1970, page 26) renditions. The test for semantic well-formedness is the basis for any semantic interpretation procedure. When the semantic procedures of a system are invoked, there are generally many interpretations possible for the input, resulting from syntactic or semantic ambiguity. The job of the semantic component is to use the meanings of the words in the input to resolve these ambiguities as much as possible. Semantic analysis generally takes one of three forms, depending on what kind of memory structure the particular system has.

Procedure oriented systems usually translate the input into some set of procedures. If the execution of these procedures fails, or if the rules for translating the input into the procedures in the first place operate unsuccessfully, a new interpretation is tried. An executing output procedure constitutes a valid test for semantic well-formedness. Indeed, the models of Woods (1968), Kellogg (1968), and Kochen (1969) do almost exactly this. Coles' (1969) system reduces the input via "production rules" to the predicate calculus which is tested for a truth value; and C. Green's (1969) QA3 uses automatic theorem proving techniques to discover the validity of the input.

The second semantic method is exactly the same as the previous method, only the output is not executable.

Thus, if the semantic transformation rules succeed on some interpretation of the input, then that interpretation is semantically well-formed. All interpretations which survive the application of these rules are valid; that is, no further processing is carried out to eliminate the remaining possibilities. Rosenbaum (1967) proposes a system such as this.

The third semantic procedure works for network oriented memories. It involves transforming the input into memory structure terminology and then searching through the memory net, via some intersection technique, for previously defined concepts which are closely related to the input. Since the system "knows" all items in its memory, this procedure is equivalent to the system understanding how the input relates to its "knowledge of the world". A valid intersection thus constitutes a test for semantic well-formedness. TLC (Quillian, 1969); PROTOSYNTHESIS III (Schwarz et al., 1970); SPINOZA II (Schank et al., 1970); and the belief models of Colby and Smith (1969); Colby, Tesler, and Enea (1969); and Abelson and Reich (1969) use this system or variants of it.

Deductive processing is also an important part of correctly interpreting natural input. To engage intelligently in natural language conversation, a well

developed logical ability is essential. It is required if the system is to give correct answers to questions, perform deductions on input, or perform any manner of logical inference. Very few models have separate, clearly defined logical procedures. The models of Coles (1969) and C. Green (1969) have the most explicit logical techniques, transforming input into the predicate calculus. Most memory net models have subset, superset, equivalence, and implication relations defined as possible links between nodes and thus have quite a logical basis. Procedure oriented systems have some logical capabilities imbedded in their rules, but do not generally have obvious deductive ability.

Output From the Model

After the model has processed its input, it should make some response, preferably one appropriate to the input. Many systems attempt to reply in natural language, but some produce other types of output.

The least natural output is a memory structure response, that is output in the model's internal representation. This response format requires very little additional processing and hence is a popular one. It also enables a person who knows the intricacies of the model to more fully understand its workings. BASEBALL

(B.F. Green et al., 1963), SAD-SAM (Lindsay, 1963), and SPINOZA II (Schank et al., 1970) respond with memory-like list structures. C. Green (1969) has his system produce output in the form of the predicate calculus.

Procedure oriented systems most often produce a procedure rather than a natural language response. In fact Kellogg's (1968) CONVERSE is the only system which actually executes its procedures to yield a natural language answer. Woods' (1968) model produces a statement in a "query" language, and Kochen's (1969) system sets up a flow chart for a procedure.

Any system trying to produce natural language responses is subject, of course, to the same type of restraints that restrict natural language input. Thus, ELIZA (Weizenbaum, 1966) and SIR (Raphael, 1968) yield natural output using paradigmatic form sentences. TLC (Quillian, 1969) also responds with paradigmatic sentence output but produces a memory structure format as well. SAMENLAQ II (Shapiro and Woodmansee, 1969) answers questions in simple relational sentences. CARPS (Charniak, 1969) and STUDENT (Bobrow, 1968) respond in single word or short phrase answers which are almost natural, but are too short to have much structure. A small generative grammar generates limited natural output for Coles' (1969)

system, and a sentence generator has been defined in PROSYNTHEX III (Schwarz et al., 1970) which gives simple English output. It is reasonable to assume that more sophisticated output techniques will soon be forthcoming to replace the somewhat crude systems that are generally in operation today.

Expanding the Model

Once a natural language model is running smoothly, the researcher inevitably becomes bored and wants to expand its scope. This could involve expanding procedures, increasing memory size and complexity, or changing any number of other items. Ideally the model should be able to extend itself and should have general facility in modifying many of its components. This is seldom the case.

Furthest from the ideal are systems which must be manually extended. In these models new words, markers, relations, and procedures must all be added by hand, so the system can be expanded only with great difficulty. This feature is particularly characteristic of many of the early systems such as BASEBALL (B.F. Green et al., 1963), STUDENT (Bobrow, 1968), and ELIZA (Weizenbaum, 1966).

A more recent approach is to define the model in such a manner that it can extend its own universe. DEACON (Craig et al., 1966) and SIR (Raphael, 1968) are set up to derive new relations from on-line discourse, although new words must be manually added. CONVERSE (Kellogg, 1968) also adds new relations automatically and, in addition, recognizes new words if they are underlined. Finally there are the systems which understand both new relations and new words from input text without any special markings. Quillian (1969) accomplishes this, for example, by assuming all words input to TLC are new in some sense. SPINOZA II (Schank et al., 1970) recognizes all words not in its dictionary and sets up new space for them. SAMENLAQ II (Shapiro and Woodmansee, 1969) not only learns from dialogue but can also generalize from examples fed to it. Becker's (1969) model derives new concepts by finding analogies to the input; and Colby and Smith's (1969) ABSI asks questions of the user, the answers to which help the system to modify its values. The models of Becker (1969) and Colby and Smith (1969) also can dynamically reassign weights to memory links to help the system learn or forget.

Testing the Model

Once the model has been fully defined, some method

of ascertaining its abilities must be sought. This usually takes the form of programming a computer simulation of the model, although some models, such as that of Becker (1969), have not been fully transferred to the computer as yet. The programming languages used are invariably symbol manipulation languages such as IPL-V, LISP, or SNOBOL.

Conclusion

From the foregoing analysis five major classes of model can be delineated. First, there are the competence models of Katz and Postal (1964) and Chomsky (1965) which are more concerned with explaining language than with modelling it. Then there are the procedure oriented models (Woods, 1968; Kellogg, 1968; Kochen, 1969) which reduce their input to procedures which can be examined or executed to determine semantic well-formedness. Models of the third type are based strongly on logic theory. They usually reduce input to an expression in the predicate calculus from which a truth value can be calculated and, if necessary, theorem proving techniques applied. Coles (1969) and C. Green (1969) have designed such systems. The fourth type of model is distinguished mainly by depending on a limited relational memory. Concepts are usually represented in this memory by

straightforward list structures consisting of words and simple relations between the words, where these relations are often semantic primitives defined as programmed subroutines. Many early systems work in this manner including SAD-SAM (Lindsay, 1963), BASE-BALL (B.F. Green et al., 1963), SIR (Raphael, 1968), STUDENT (Bobrow, 1968), and ELIZA (Weizenbaum, 1966).

Models in the fifth class are the direct successors to those in the fourth and often have much the same sort of design. As in fourth category models, concepts are represented in memories consisting of words and links, and procedures are used to translate input into memory. Their main differences lie in the sophistication of their memory structures (which are often highly nested), the uniformity of their methods for determining semantic well-formedness (often through intersection techniques), and the generality of input and subject matter which they allow. Examples of models belonging to the fifth class are those devised by Craig et al. (1966), Shapiro and Woodmansee (1969), Colby and Smith (1969), Abelson and Reich (1969), Becker (1969), Schank et al. (1970), Schwarz et al. (1970), and Quillian (1969). Quillian's TLC is particularly important since much of MUSE is based directly on it. Chapter II is devoted to a thorough study of TLC.

CHAPTER II

THE TEACHABLE LANGUAGE COMPREHENDER

Introduction

This chapter presents a detailed analysis of a recent natural language model - Quillian's Teachable Language Comprehender (TLC) (Quillian, 1969). This is undertaken in the hope that the study of the workings of a particular model will help to further illustrate some of the problems associated with natural language modelling. TLC has been specifically chosen since understanding MUSE depends on understanding TLC.

TLC is the latest version in a long series of models, devised by M. Ross Quillian, which rely heavily on semantic processing (Quillian, 1966, 1967, 1968). It is a computer model (programmed in BBN-LISP) that is designed to comprehend natural input, to learn from on-line discourse, and to respond with natural language output. The basis for TLC is its semantic memory.

The Memory Structure of TLC

TLC's semantic memory is a completely abstract network of units and properties. Each unit in the memory represents some concept which the system understands; and, although many units are linked to English

words in a dictionary, no unit necessarily has to correspond to an English word. Attached to a unit is a pointer to another unit, its superset, and possibly pointers to a number of subproperties. These properties modify the superset in such a manner as to correctly code the concept that the unit represents.

A property is an attribute-value pair symbolizing a predicate to the system. The notion of attribute-value has been expanded to include preposition-object (e.g. "down street") and verb-object (e.g. "chase car"), as well as the traditional meaning of the term (e.g. "speed fast"). Each property has pointers to units representing its attribute and value and, if needed, pointers to further subproperties modifying the original property. Since each subproperty can have further subproperties, arbitrarily complex predicates can be represented in the system. Also appended to each property is a pointer to a set of "form tests" to be used in the syntactic processing of the input. These form tests are not part of the semantic memory, so they will not be explained in further detail at this point. In addition, Quillian indicates that the memory format provides for the quantification of units and for the grouping of units into sets, but he gives no details of these provisions.

For example, a dog named Ruff who chases cars could be represented as shown in Figure 2.1. The square

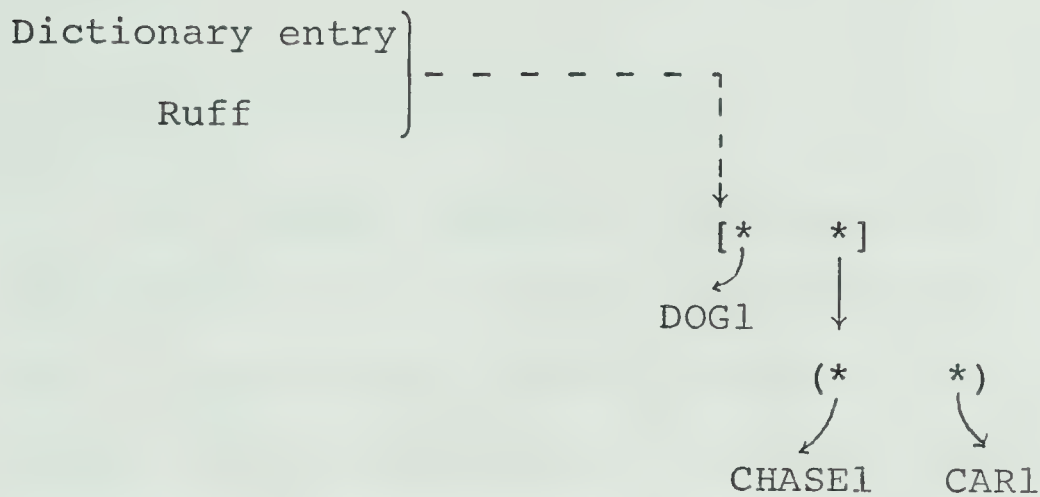


Figure 2.1 The memory concept for RUFF1

brackets represent units, the round brackets are properties, the capitalized words are dictionary entries. The numbers after the words represent the particular meaning involved (e.g. there might be another meaning for Ruff called RUFF2) and do not exist in the actual memory. In the example above, Ruff is the dictionary entry for the corresponding unit RUFF1 in memory. From now on, pointers to units and words will be represented by their English equivalents in the above notation. It can be seen from Figure 2.1 that the word Ruff is entered in a dictionary and is linked to a unit in memory (RUFF1). This unit has the superset DOG1 and a subproperty

consisting of attribute CHASE1 and value CAR1 with no further subproperties. The property (CHASE1 CAR1) modifies the superset DOG1 to yield the memory's concept of RUFF1.

Interpreting the Input to TLC

The system comprehends English input (which is unrestricted, in principle) by translating the input into an appropriate coding in memory format which can then be added to the memory. In this manner the model understands the input, and at the same time builds up its "knowledge of the world". Syntactic, semantic, and logical processing are all combined in this single interpretation process.

Before proceeding further, an example is given which will aid in understanding the way that the model works. Assume that the definition of RUFF1 above is coded in memory, and the sentence "Ruff chases Volkswagen" is read into the system using the command READ (RUFF CHASE S VOLKSWAGEN). The single letter "S", written separately in the input, is the present tense suffix for the verb CHASE. All suffixes in TLC are written separately so that the model can avoid the necessity of complicated syntactic processing to discover root words. The input is taken to be a completely new

concept, at least for the present, and new empty units are set up to represent the new meanings of Ruff, Chase, and Volkswagen. To avoid confusion these empty units will be called u_1 , u_2 , and u_3 respectively. The job of TLC is to associate with each u_1 , u_2 , and u_3 an appropriate superset and suitable subproperties.

To accomplish this, a set of "candidate" units for each word is prepared. A candidate unit for a word is a possible meaning that the word might have in the input. Thus, the memory might contain three candidate meanings for Ruff: RUFF1, a dog (as in Figure 2.1); RUFF2, a band around the neck; or RUFF3, to trump at bridge. Similarly there might be several meanings for chase and Volkswagen, but for the present example it will be assumed that each of these words has only one definition in memory: CHASE1, meaning to pursue down the street; and VOLKSWAGEN1, the car.

TLC must now choose from the candidate meanings the sense of each word which is appropriate in the present context. This appropriate meaning will then form the superset for the new empty unit for that word. In the example the system should choose RUFF1, a dog, as an appropriate meaning for u_1 , and, of course, choose the units CHASE1 and VOLKSWAGEN1 as appropriate for u_2 and u_3 .

Once the supersets for u_1 , u_2 , and u_3 are filled in, reasonable subproperties must also be found. TLC achieves this by adapting properties which are already in memory; in fact, by adapting the properties of the newly discovered supersets. In the example the properties of RUFF1, the dog, would be adapted to fit the context of the input. The new properties would then be added to u_1 . More details of how TLC adds new properties and new supersets to u_1 , u_2 , and u_3 will be given after a fuller theoretical explanation of this process.

TLC finds the supersets and properties of the new units using a single procedure: a parallel search through memory. The search attempts to find "semantically related" candidate units for different words in the input, on the theory that any two candidates which can be related in memory are more likely to be the "correct" choices for their respective words in the input. Thus, TLC uses its "knowledge of the world" to decipher the context of the input sentence.

Before the search algorithm can be understood, however, several terms must be defined. Two units are "semantically related" if one unit is "semantically identifiable" with the attribute or value of a property of the other unit. A unit and a part of a property

(or any two units for that matter) are "semantically identifiable" if they have a "superset intersection" in memory. Two units have a "superset intersection" in memory

"(1) If the two units are the same unit.

(2) If one unit is the superset of the other, or the superset of the superset of the other, or the superset of that, etc.

In this case, we will say that one unit lies directly on the "superset chain" of the other.

(3) If the two units' superset chains merge at some point."

(Quillian, 1969, p.467).

Observe the difference between semantic relationship and semantic identity: semantic relationship involves the intersection of a part of a property with a unit, whereas semantic identity just involves the intersection of two units. Also, note that in TLC the length of a superset chain has been arbitrarily limited to three supersets so that the model will not have to look an unreasonable distance for an intersection.

The search looks at the properties of all the candidate units, starting with the most recently defined. It expands superset chains from the attribute and value

of each property of each candidate, as well as building superset chains from all the candidates themselves. By expanding each chain only one link at a time before proceeding to the next chain, the search effectively simulates parallel processing. Eventually a superset intersection occurs between a part of a property of one candidate and one of the candidate units of the other words in the input. These two candidates are thus semantically related and are tentatively chosen as the correct choices for the supersets of the corresponding empty units, pending satisfaction of a form test between the candidates. Form tests will be explained shortly, but first the processing of the example to this point should be examined.

TLC would start expanding the superset chains from the attributes and values of the properties of RUFF1, RUFF2, RUFF3, CHASE1, and VOLKSWAGEN1. In addition chains would be built up from each of the candidates. Since RUFF2, a band about the neck, and RUFF3, to trump at bridge, are not closely related to the input, the first superset intersection is likely to occur between a property of RUFF1, a dog, and one of CHASE1 or VOLKSWAGEN1. The memory definition RUFF1, given in Figure 2.1, contains one property, (CHASE1 CAR1), and

the candidate for the second word of the input is CHASE1. Hence, the attribute of a property of RUFF1 (namely CHASE1) has a superset intersection with a candidate unit for another word of the input (also CHASE1), since they are the same unit. Thus, RUFF1 and CHASE1 are semantically related. Similarly, RUFF1 and VOLKSWAGEN1 are semantically related, since CAR1 is both the value of a property of RUFF1 and on the superset chain of VOLKSWAGEN1. Therefore, RUFF1, CHASE1, and VOLKSWAGEN1 are tentative choices as the supersets of u1, u2, and u3.

To make the identification permanent, each pair of semantically related candidates must satisfy a form test. Form tests are the syntactic element of TLC. Each property of the memory has associated with it two sets of form tests, one set to be used if the attribute of the property is semantically identified with another unit, and the other set to be used if the value is so identified. The rules in a form test look for a valid syntactic relationship between the two candidate units that the test is examining. Such a relationship could be the existence of a preposition between the words (e.g. "for" in "lawyer for client"), the presence of an "'s" after the first word (as in "lawyer's client"), or any number of other possibilities. If all form tests

fail for any pair of candidates, TLC asks the monitor (program user) for a new form test which will apply. If no new test is forthcoming, TLC assumes that the current intersection is not valid, and it proceeds with its search.

In the example above, since the property (CHASE1 CAR1) has been used to semantically relate RUFF1 to CHASE1, form tests associated with (CHASE1 CAR1) would be employed on RUFF1 and CHASE1. Because RUFF1 and CHASE1 are related through the attribute of this property, form tests of the first kind would be tried. One such form test might test whether the word in the input corresponding to CHASE1 follows the word corresponding to RUFF1. Similarly, a form test of the second kind would check the relationship between RUFF1 and VOLKSWAGEN1. If the form tests both succeed, RUFF1, CHASE1, and VOLKSWAGEN1 can be permanently added as the supersets of u1, u2, and u3.

Discovering the supersets of the empty units also gives a method for adding new properties. These properties are actually adaptations of old properties in memory, specifically old properties which have been employed in getting the new supersets. Each such old property is copied from the unit with which it is associated and placed under the corresponding new unit. If the attribute or value of this property has been found to be semantically

identifiable with the candidate of another word during the intersection process, then this attribute or value is replaced by the new unit representing that word. Otherwise the property is left unchanged.

In the example the only property which has been used in finding the new supersets is the property (CHASE1 CAR1) which is appended to the unit RUFF1. Hence, the associated new unit u1 is given the property (CHASE1 CAR1). Since the attribute CHASE1 and the value CAR1 have been semantically identified with candidates CHASE1 and VOLKSWAGEN1 respectively, they must both be replaced by the corresponding new units. In this case CHASE1 is replaced by u2 and CAR1 by u3. Thus, the new unit u1 has the new property (u2 u3) attached to it. Note that RUFF1 still has (CHASE1 CAR1) as a property, since the copying and modifying of properties in no way changes old concepts in memory.

After TLC has set up new units and new properties, it creates, if desired by the monitor, new dictionary entries by associating the new units with the proper words in the input. Therefore, Ruff, Chase, and Volkswagen could be the dictionary entries associated with units u1, u2, and u3 (which for consistency of notation will now be termed RUFF4, CHASE2, and VOLKSWAGEN2). Figure 2.2 shows the new concept which TLC has just produced.

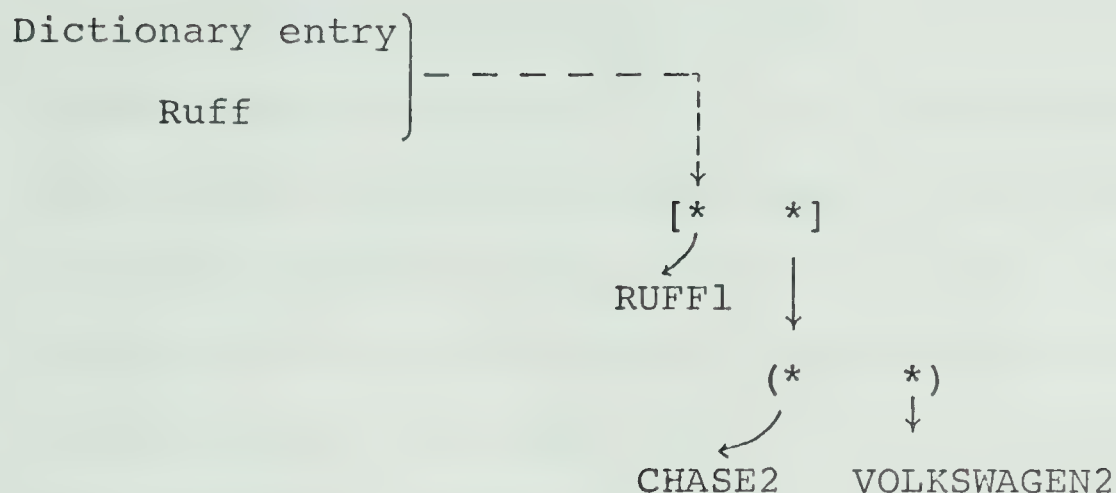


Figure 2.2 The memory concept for RUFF4

The final action which TLC undertakes is to respond in fixed format (paradigmatic) English. This natural language response indicates to the monitor how TLC has interpreted the input. Just to be sure that the monitor understands what is going on, however, the program also produces a nested structure corresponding to its internal representation of the input. In the example the output might look something to like

```
((RUFF (CHASE (VOLKSWAGEN))))
```

HERE WE ARE CONCERNED WITH RUFF WHO CHASES A VOLKSWAGEN

Figure 2.3 Output from TLC

An Evaluation of TLC

The Teachable Language Comprehender is a very elegant model of language behavior. Its procedures are based

on what little is understood about the one comprehensive natural language processor so far in existence: the human brain. Thus, TLC relies heavily on intersection "fans" spreading out from concepts in memory in a breadth-first procedure similar to the parallel processing which is theorized in the human brain. A second feature somewhat analogous to the human brain is TLC's highly inter-related, content addressable semantic memory which is responsible for many of TLC's success. Both features differ markedly from the serial processing and location addressable memory that have traditionally been associated with computers.

The essence of TLC is its simplicity. The memory structure consists of only two entities, properties and units, and yet it is able to represent complicated nested concepts to any level of recursion. The superset connection between units allows impressive deductive processing to be undertaken, and the broad definition of the properties allows relations of remarkable richness to link the memory together. The memory is also well suited for the easy translation of natural input into its format.

No less an achievement is the procedure for interpreting the input. Using a single algorithm, TLC not only comprehends natural input, deciphering many of its ambiguities, but it is in a position to use the information

contained in the input to properly update its "knowledge of the world". In fact, comprehension in TLC is defined to be the translation of the input into memory structure format. By coding all text in the same constructions as those of memory, the model also eliminates the need for special procedures to process anaphoric references (references to something already stated in text).

Among the weaknesses of TLC is its very emphasis on semantics which, while good in itself, results in syntactic processing being reduced to a few form tests. Even these are somewhat crude, leading Quillian to propose using a transformational syntactic approach similar to that programmed by Dewar et al. (1969). Perhaps, too, syntax could be used to help eliminate the multiplicity of candidate units before the time consuming search routine goes into operation. Syntax could also be invaluable in discovering which words of input are likely to be related so that TLC could concentrate almost exclusively on looking for semantic intersections among these syntactically related words. Finally, syntactic procedures to handle suffixes would eliminate the artificial separation of suffixes from the words with which they are associated.

Quillian points out that the intersection techniques need to be refined to some extent, as well. Procedures

which allow the system to correctly interpret "enemy's lawyer", for example, lead it to falsely interpret "lawyer's enemy" due to overpowerful intersection techniques. On the other hand, many intersections (such as intersections between parts of separate properties) which are potentially useful in what Quillian calls "indirect comprehension" (Quillian, 1969, p.467) are ignored. TLC also has difficulty comprehending totally new concepts, since no completely new properties can be added, only adapted properties from old concepts.

Adding a new unit for each word of input means that TLC builds up its memory much too rapidly. Procedures should be devised which allow the system to "forget" most of its input. In addition, the attaching of each new unit as the subset of an old unit allows superset chains of extreme length to be developed. The forgetting routines should also be able to amalgamate or eliminate many of these units.

TLC has no explicit question answering techniques. As will be seen in the next chapter, some implicit question answering is possible as a by-product of the comprehension process, but there are no direct procedures for responding to queries. Neither is TLC designed to perform "motor actions" on the interpretation it has

given the input. For example it would be useful to have the system not only be able to code the command "Halt processing" in its memory format, but also be able to actually halt after so coding. At present, the only action that the system can undertake is READ, which precedes all input and just tells the system to perform its comprehension process.

The final fault which will be mentioned here is that TLC is quite hard to use. The monitor has to be trained to know exactly how to state input, how to understand the response of the system well enough to be able to see if it is performing properly, and whether to add the newly interpreted input to memory. He must also know what to do when TLC requests a new form test. As the system develops beyond the experimental stage, these problems will likely disappear.

MUSE has been designed to overcome some of the shortcomings of TLC. Although less sophisticated than Quillian's model in some areas, MUSE is equal or even superior to TLC in others. Chapter III details the theory behind MUSE.

CHAPTER III

FORMULATION OF MUSE

Introduction

MUSE is a natural language model which understands simple written English input by transforming it into a grammatically correct and semantically well-formed internal representation, called the model's "interpretation" of the input. Any motor actions required are then performed, if possible, before the model asks for more text to be input. Some portions of this understanding process are borrowed from other natural language models, notably TLC, and some parts are original. This chapter describes the design of MUSE, although many details of the implementation are left to Chapter IV.

MUSE is based directly on TLC, so that most of the terms used here have the same definitions as in Chapter II. MUSE attempts to overcome many of the faults of TLC while still retaining the semantic orientation of Quillian's model. Syntax in MUSE is more sophisticated than in TLC and enables the system to eliminate many useless candidate units before the complicated semantic processes are implemented. The syntactic component also helps to direct the semantic

component; that is syntax is used to discover what words of the input are most likely to be semantically related. As well, syntax aids in the derivation of new properties, rather than just checking old properties, as is done in TLC.

Instead of adding new units to memory for every word of the input, MUSE tries to modify old units to encompass the input concept. In fact, the only way that new units can be added is by specially marking the input words. Thus, MUSE avoids the problems of excessive memory size and extraneous supersets. MUSE also allows for "actions" (pre-programmed executive routines) to be linked with each unit. These actions can be specialized syntactic and semantic procedures to act on the input during the interpretation process or motor actions such as "halt" which act after the input has been interpreted. Finally, limited question answering facilities have been incorporated into MUSE.

MUSE is probably not as flexible as TLC, mostly due to the limitations of the syntactic processes. Thus, the breadth-first intersection order of TLC is abandoned to a large extent in favor of more localized syntax directed searches. In addition, MUSE eliminates suffixes, tenses, plurals and the like so that root words

can be more easily discovered. Also the format which must be used to communicate with MUSE is probably more inflexible than that of TLC. The monitor must know quite a lot about the workings of the system before he can use it, especially if he is defining new words, or editing MUSE's memory.

The Memory Structure of Muse

MUSE has a memory structure composed of three components: a dictionary, a semantic memory, and a syntax file. The dictionary consists of English words with pointers to corresponding units in the semantic memory. A word can be any expression comprised of English letters (A B C ... Z) with no blanks. Because many words can have more than one meaning, it is possible for a word in the dictionary to have pointers to more than one unit in the semantic memory. Of course, concepts in the semantic memory do not have to be stateable in English, so that not all units have to have a corresponding dictionary word. However, in the examples which have been run so far, all units do have corresponding dictionary words.

The semantic memory consists of a file containing units and a file containing properties. These units and properties can be considered to be conceptually identical

to those of TLC. There are some differences, however. A unit has a pointer to its superset and pointers to its subproperties as in TLC; but in addition it can have pointers to any actions to be undertaken when that unit is encountered at various stages of processing. A part of speech marker, to be used in syntactic analysis, is also attached to each unit. The properties are unchanged from TLC. Note that for the sake of simplicity no allowances have been made in the memory format for the quantification of units or for units which have been grouped into sets.

The third portion of MUSE's memory structure is a file containing part of speech combinations and corresponding phrase markers (sometimes called "parse trees"). This is the syntactic component of MUSE and takes the place of TLC's form tests. All valid parse trees (at least as far as MUSE is concerned) can be extracted from this syntax file. A valid parse tree in MUSE must satisfy certain special conditions to enable correct processing to take place. The tree must have exactly two branches from all nodes except terminal nodes; it must define all adjectives and adverbs in adjective and adverb phrases which must follow the word they are modifying, and the first branch of which must end in a "null" node; and the tree must define a branch ending in a null node to

stand for the implied subject of any imperative sentences. The last two conditions are to allow the model to properly perform specific actions, and a further explanation is given later in the chapter. By storing phrase markers in a file instead of defining a parse procedure, many difficulties of implementation and excess processing time are avoided, although eventually a programmed parsing routine (such as those used in Coles, 1969 and Tharp and Krulee, 1969) should be added to make MUSE more general.

Processing the Input to Muse

A flow chart outlining the operation of MUSE is given in Figure 3.1. The present section explains this flow diagram with the help of a familiar example. Once more it is assumed that the model knows three meanings of Ruff, one of chase, and one of Volkswagen, the famous RUFF1 (a dog), RUFF2 (a band around the neck), RUFF3 (to trump at bridge), CHASE1, and VOLKSWAGEN1 of the last chapter. Again suppose that "Ruff chases Volkswagen" is the input sentence to be understood by the model.

The input to the model must satisfy several restrictions before it can be successfully processed. The input must not contain adjectives, adverbs, or prepositional phrases which modify other adjectives, adverbs, or prepositional phrases; but adjectives, adverbs, and

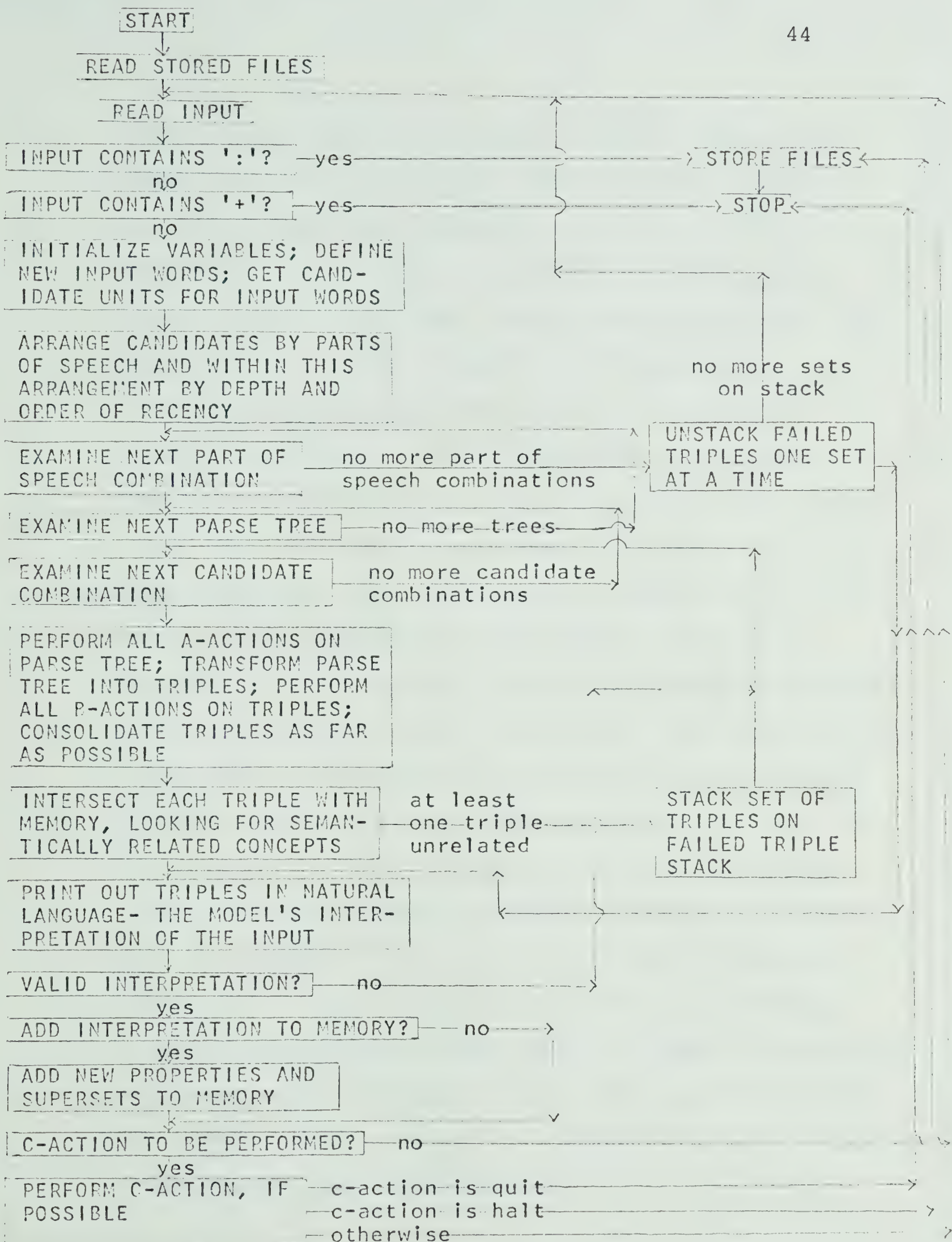


Figure 3.1 Flow diagram for MUSE

prepositional phrases may modify nouns or verbs. Conjunctions between words (e.g. "dog and cat and mouse") are not allowed, but conjunctions between clauses are acceptable. In addition, quantification and negation cannot be handled by MUSE. Once a word in the input has been defined, it must always take the same form in all future references, since MUSE has no facilities for recognizing plurals, tenses, agreements or the like. Thus, for simplicity all nouns are assumed to be singular (e.g. "dog" not "dogs"), and verbs to be infinitives (e.g. "be" instead of "is", "was", etc.). Moreover, if a new word is being introduced in the input, it must have appended to it in brackets its part of speech and pointers to all actions which apply to the word. All words in the input must be separated by blanks, and no punctuation is allowed except a period or question mark at the end. No words may be capitalized, and all must satisfy restrictions placed on dictionary entries. Finally, all input must be processable by both the syntactic and semantic components of MUSE, so that, for example, no sentences will be understandable unless they have appropriate phrase markers stored in the syntax file. The input sentence in the example would be typed "ruff chase volkswagen", which satisfies all input restrictions.

MUSE initiates processing by notifying the monitor that it has started by typing the sentence "GOOD MORNING. MY NAME IS MUSE.". The model then reads stored files containing the dictionary, the units, the properties, and the syntax for the system. MUSE is now ready to accept input so it states "PLEASE TYPE YOUR INPUT." and waits for a response. It is at this point that "ruff chase volkswagen." is typed. Upon receiving the sentence to be processed, MUSE checks to see if the input contains "+" or ":", the fast exits from MUSE, so called because they tell MUSE to immediately stop processing. A "+" indicates an immediate stop without storing any memory files, whereas a ":" is used when it is desired to stop MUSE after storing its files. In either case MUSE prints out "THANKS FOR THE CONVERSATION." and stops in the appropriate manner.

If the input does not require a fast exit, MUSE initializes a number of variables and proceeds to attempt to "interpret" the input. As mentioned in the introduction, an interpretation of the input is a syntactically correct and semantically well-formed internal representation of the input. A "valid" interpretation is an interpretation which the monitor has judged to be correct. When trying to discover a valid interpretation,

the first thing the model does is read each input word, delineated by blanks, until it encounters a "." or "?", indicating the end of the input. MUSE then checks for new words which, it will be recalled, are marked with brackets containing a part of speech symbol and action pointers. Any new words found are added to the dictionary, either as another meaning of an old word, or, if necessary, as an entirely new word. The new dictionary entry contains a pointer to the new unit which is defined for each new word. This unit is filled in with the part of speech symbol and action pointers contained in the brackets, and is also initially given a pointer to the null superset and the null property. Properties and supersets are later discovered from input text. Note that since new units are added to the end of the unit file, then the higher the unit number the newer the unit (e.g. RUFF3 is newer than RUFF2). All new units are flagged for the duration of the processing of the current input sentence. This is done so that no other candidate units will be tried for the new word, and so that the actions associated with the new unit will be suspended long enough to ensure proper interpretation of the input (i.e. if the action associated with a unit told MUSE to delete the unit wherever it appeared, then the unit would be deleted before it could be added to memory unless the actions were suspended the

first time through). Of course, in the "ruff chase volkswagen." example no new words have been marked, so most of this processing is avoided altogether.

MUSE now proceeds to discover the candidate units for each word in the input. The candidate units for a word are all units associated with that word in the dictionary (called "first level" candidates), plus all "subsets" of these units. UNIT1 is a subset of UNIT2 if UNIT2 lies on the superset chain of UNIT1. Note that there is no limit to how deep the model goes when looking for subsets - it keeps going until all subsets are found. This feature becomes quite important in question answering, as will be presently seen. In the example the dictionary units associated with "ruff" are the units RUFF1, RUFF2, and RUFF3; and since none of these units has a subset, these are the only candidates. Similarly the units CHASE1 and VOLKSWAGEN1 are the only candidates for the input words "chase" and "volkswagen".

MUSE is now ready to choose "combinations" of these candidates and do further processing on each such combination. A combination of candidate units is a vector of units representing one possible choice for the meaning of each input word. It is desirable to have some sort of order in which to examine the candidate combinations,

preferably an order that will yield the "correct" combination (i.e. a combination which results in a valid interpretation) as quickly as possible. Toward this end MUSE orders the candidates for each word, grouping the candidates first according to their parts of speech. The candidates in each part of speech group are then arranged according to how deeply they are removed from the first level candidates, the "shallowest" units coming first. Finally, any candidates in the same part of speech group and on the same level, are ordered so that the most recently defined come first. Thus, the candidates for "ruff" in the example would be arranged RUFF3, to trump, as the only verb; followed by RUFF2, a band around the neck, and RUFF1, the dog, as the nouns which are arranged in order of recency since they are on the same level in the semantic memory. CHASE1 and VOLKSWAGEN1, being the only candidates for the second and third input words, need no ordering. Thus, there are three possible combinations of candidates: RUFF3,CHASE1,VOLKSWAGEN1; RUFF2,CHASE1,VOLKSWAGEN1; and RUFF1,CHASE1,VOLKSWAGEN1.

The actual choosing of the candidate combinations is now accomplished with the help of a generator which enumerates all possible combinations of integers in some vector, as long as the integers are less than the integers

in some maximum vector. The generator starts with a vector of 1's and, by increasing the leftmost columns of the vector first, produces new combinations one at a time until the maximum vector has been reached. For example, if the maximum vector is 2 3 1, the following vectors would be produced in order: 1 1 1; 2 1 1; 1 2 1; 2 2 1; 1 3 1; 2 3 1.

It is clear that the same sort of generator can be used to generate indices to examine appropriate combinations of candidates, one at a time. In fact, two such generators have been defined, one to produce, one by one, all "part of speech combinations" and the other to give all possible combinations of candidates within each "part of speech combination" (a part of speech combination is a vector of parts of speech of the input).

It now becomes clear why the candidates are ordered with part of speech first - to enable the part of speech generator to examine the part of speech combinations without redundancy. This saves a good deal of processing time, although it does detract a little from the desired depth-first order of candidate examination. In practice, however, this objection turns out not to be too serious, since usually most candidate units for a word are the same part of speech, thus allowing depth-first ordering to once

again become dominant. In the "ruff chase volkswagen" example there would be two part of speech combinations: verb,verb,noun and noun,verb,noun. Within verb,verb,noun there is only one combination of candidate units, RUFF3,CHASE1,VOLKSWAGEN1; but within noun,verb,noun there are two candidate combinations: RUFF2,CHASE1,VOLKSWAGEN1 and RUFF1,CHASE1,VOLKSWAGEN1.

Syntactic Processing of the Input

MUSE now generates the first part of speech combination and searches through the syntax file for associated parse trees. If no parse trees are discovered, the model rejects the current part of speech combination (along with, of course, all candidate combinations which are subsidiary to that part of speech combination), and generates a new part of speech combination. If one or more parse trees are found, the first is taken and processed until it either yields a valid interpretation or it is rejected. If it is rejected, the next is taken, and so on until all parse trees have been tried (whereupon a new part of speech combination is generated) or until a correct parse tree has been discovered. The parse tree being examined at any one time is called the "current" parse tree.

In the example the verb,verb,noun part of speech combination probably would have no parse associated with

it, so would be rejected immediately. In this case, the candidate combination RUFF3,CHASE1,VOLKSWAGEN1 is no longer considered a possibility for the meaning of the input. On the other hand, there likely would be a parse tree associated with noun,verb,noun (such as the tree of Figure 3.2), and this would become the

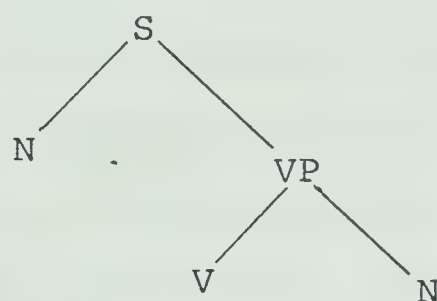


Figure 3.2 A parse tree

current parse tree. Thus, the syntax look-up has already eliminated one candidate combination for the meaning of the input, although two more remain: RUFF2,CHASE1,VOLKSWAGEN1 and RUFF1,CHASE1,VOLKSWAGEN1. MUSE must continue processing in order to choose between these two candidate combinations.

After obtaining a parse tree, MUSE calls in the candidate combination generator to produce, in order, the candidate combinations which can be passed using the current parse tree. Each candidate combination, as it is produced, becomes the "current" candidate combination and is further processed until it is successfully inter-

preted or until a failure occurs somewhere. If a failure occurs, the next candidate combination becomes current and is tested. When no more candidate combinations can be produced, the model returns for a new parse tree. Once a current candidate combination has been produced, the units comprising the combination become the terminal elements for the current parse tree. In the example, RUFF2,CHASE1,VOLKSWAGEN1 would become the current candidate combination, and the parse tree would be filled in as in Figure 3.3. If this candidate combination were to fail, RUFF1,CHASE1,VOLKSWAGEN1 would be tried.

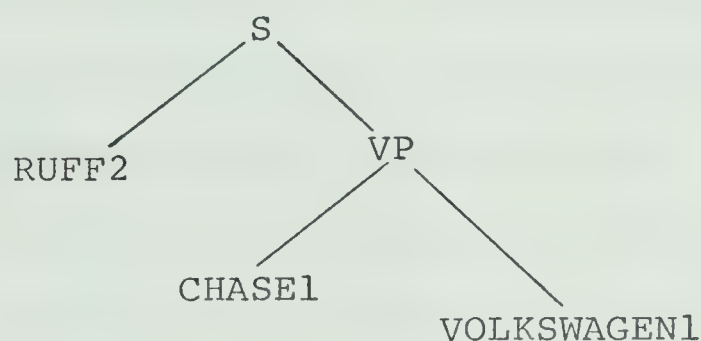


Figure 3.3 Parse tree filled with candidate combination

MUSE now performs any "A-actions" which are associated with the candidate units in the current candidate combination. Recall that an action is a pre-programmed executive routine which is linked to a unit. There are actually three types of actions in MUSE (A-actions,

B-actions, C-actions) which are applicable at different stages of the processing of an input sentence. Any unit can have at most one of each type of action associated with it. A-actions are actions which are applied to the terminal units of the current parse tree at this point. These actions include one that deletes the associated candidate unit from the current parse tree, such as is done with articles like "the" and "a" and subordinate conjunctions like "that". Units with this action have syntactic significance to MUSE (since the units were used in obtaining the current parse tree), but no semantic significance. Another A-action adds the unit SELF1 to the parse tree as the subject of a verb which has no object in the current parse. This allows MUSE to process imperative sentences. The final A-action inserts the superset of an adjective or adverb into the current parse tree, thus enabling the model to handle attribute-value pairs which modify the word with which they are associated. The attribute is the superset of the adjective or adverb, and the value is the adjective or adverb itself (e.g. color brown, speed fast, etc.).

For both the second and third A-actions, the parse tree must be specially designed to allow the addition of the units in the appropriate place. This is accomplished

through the use of branches in the parse tree that end in the "null" units. These null units are then filled in with the proper units derived by the A-action. An example of the third A-action, adding a superset to an adjective or adverb phrase, is now given. Figure 3.4(a) shows the parse tree for the input phrase "brown dog" (with associated units BROWN1 and DOG1), and Figure 3.4(b) shows what happens when the A-action of the third type which is associated with BROWN1 is applied to the parse tree.

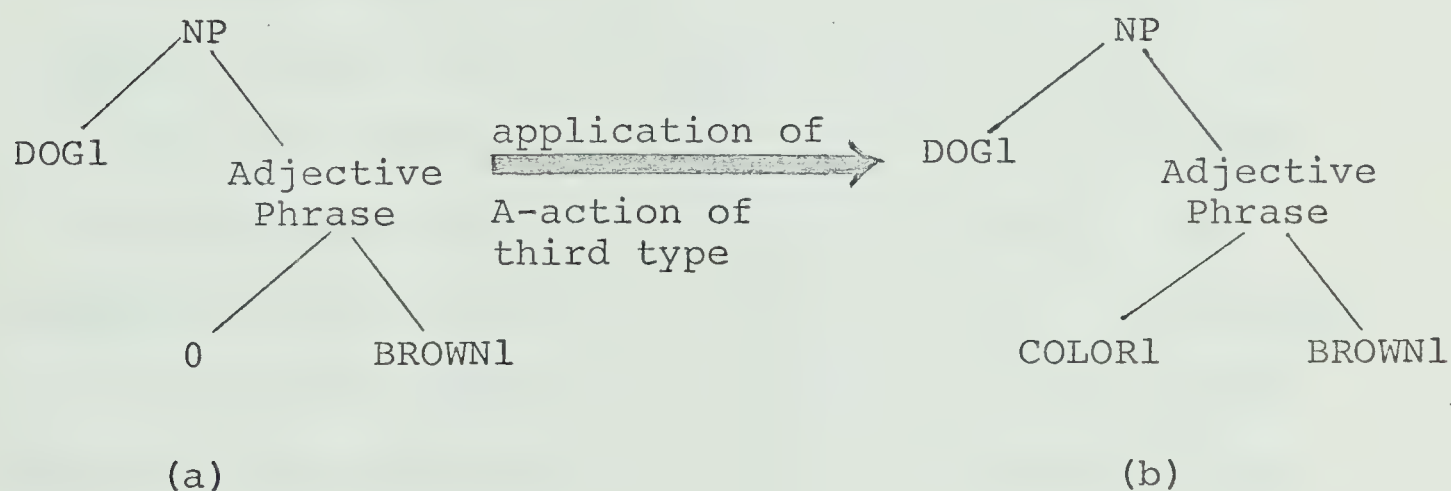


Figure 3.4 A-action of the third type

The figure can be explained as follows. Assume that MUSE is looking at the current candidate combination BROWN1,DOG1 and is processing this combination from left to right looking for A-actions. It finds an A-action of the third type linked to BROWN1, so it turns to the current parse tree (Figure 3.4(a)) and finds that BROWN1 is dominated

by an adjective phrase which has a null node. MUSE finds the superset of BROWN1, say COLOR1, and replaces the null node by the unit COLOR1 (see tree in Figure 3.4(b)), thus completing the action. In the "ruff chase volkswagen." example, no A-actions are associated with any of the units in the current candidate combination, so the current parse tree remains unchanged (still Figure 3.3).

Once the A-actions have been performed, MUSE attempts to convert the current parse tree into "relational triple notation". This is similar to the triple notation defined in Schwarz et al. (1970), and involves amalgamating the terminal units of the tree into triples which are syntactically related. The essential idea is to associate the nouns in the parse tree with their modifying verb phrases, prepositional phrases, and adjective phrases; and to associate the verbs in the tree with their modifying prepositional phrases and adverb phrases. Figure 3.4(b) shows the general structure of a unit and its modifying phrase - the unit (DOG1) and the modifying phrase (Adjective Phrase) are both on the same tree level, and the components of the phrase (COLOR1, BROWN1) are on the next level. Thus, a relational triple is defined to be an

ordered triple of terminal units U1, U2, and U3 of a parse tree such that U1 is a noun or a verb on the same tree level as a modifying adjective phrase, adverb phrase, prepositional phrase, or verb phrase; and U2 and U3 are the terminal units of the two branches leading from this modifying phrase.

Breaking a parse tree into relational triple notation consists of finding all the relational triples in the parse tree and placing them one after another in a "vector" of triples. All parse trees which are still active at this point in MUSE's processing should be reducible to relational triple notation, since both the input and the parse trees themselves have been restricted especially to allow this. For example the relational triple notation for the sentence "rover chase green volkswagen down street." would be (ROVER1 CHASE1 VOLKSWAGEN2) (CHASE1 DOWN1 STREET1) (VOLKSWAGEN2 COLOR1 GREEN1). The relational triple notation for the current parse tree in the "ruff chase volkswagen." example consists of but one triple: (RUFF2 CHASE1 VOLKSWAGEN1).

The model must now perform all "B-actions". These actions, associated with the units of the current candidate combination, perform various procedures on the newly discovered relational triples. There are two B-actions which are necessary in MUSE, the first because



of the way the verb "to be" is defined in the model, the second because of the manner in which prepositions are specified in the semantic memory. In MUSE the input sentence "A be C" usually means that C is the superset of A, i.e. the role of the verb "to be" is usually to denote superset relationship between units. To indicate to the model that a superset relationship obtains between the units of the triple, rather than the ordinary unit modified by a phrase, the verb "to be" is replaced by the null unit (i.e. "0") wherever it occurs in the triples. The remaining two units are considered by MUSE to have a superset relationship between them. Thus, the first B-action is executed on all units in memory that define a superset relationship (e.g. "to be"), and it results in the unit being replaced by the null unit everywhere in the triples. The triple (DOG1 BE1 ANIMAL1) would, for example, be reduced to (DOG1 0 ANIMAL1) if BE1 had a B-action of the first type associated with it.

The second B-action is needed because of the way prepositions are defined in MUSE. A preposition is considered to have no subsets or properties, its meaning being determined by its superset. Thus, the preposition OF1 in the triple (CAR1 OF1 JOHN1) might have superset HAVE1, indicating that in the present context OF1 means HAVE1. The second B-action when

applied to a unit in a triple, replaces that unit by its superset. In the current example this would yield (CAR1 HAVE1 JOHN1) which is the opposite of the intended meaning. Therefore, the second B-action also reverses the units surrounding the action unit in the triple to yield (JOHN1 HAVE1 CAR1), the correct interpretation in the present circumstances. If the superset is the null unit, the units are still reversed, but the action unit is replaced by the null unit. For example, the unit OF2 in (COLOR1 OF2 DOG1) might be defined as having no superset in memory, but still have the second B-action. When the B-action is applied to the triple, it becomes (DOG1 O COLOR1). This is an example of a second type of two unit triple, one that does not define a subset-superset relationship. "Consolidation" routines (explained in the next section) help MUSE to amalgamate such two unit triples with other triples of the input, and hence eliminate any ambiguity. Thus, the second B-action is able to handle prepositions with null supersets. Of course, this action does not have to be associated with all prepositions, only with those such as "of" or "for" which define an inverted relationship between words. Any preposition without the second B-action, or which has the second B-action but does not appear in the middle of the triple, is left alone.

The B-actions leave many loose ends untied. What happens to a triple which has only one unit or has all units the same? What does the system do when it encounters a "to be" verb which does not define a superset relation at all, such as in the sentence "dog be brown."? How does the model handle statements like "chase be pursue down street." so that the prepositional phrase "down street" will modify the verb "chase", not the verb "pursue"? These and other problems are solved by a set of routines which "consolidate" the triples as much as possible. The consolidation routines are ad hoc procedures derived from a few examples, but it is hoped that they are generalizable to a reasonably large set of words. Only testing MUSE for many sentences will determine how comprehensive the routines really are.

During the execution of the B-actions, it is possible that some triples will have been reduced to one unit or, perhaps, no units. It is also possible that some triples will have been defined whose units are all identical. In either case, it is assumed that the triple contains no real information. Thus, the first consolidation rule is to delete all triples which contain no more than one distinct unit.

As mentioned above, problems arise when the verb "to be" does not imply a superset relation, but merely



a linking between an adjective and a noun. The triple notation representation of the sentence "dog be brown." would be (DOG1 BE1 COLOR1) (BE1 COLOR1 BROWN1), the unit COLOR1 resulting from the A-action which adds the superset COLOR1 to the adjective BROWN1. The first B-action associated with BE1 would reduce the triples to (DOG1 O COLOR1) (O COLOR1 BROWN1). Now, it is clear that these two triples are closely related and, in fact, can be amalgamated to the single triple (DOG1 COLOR1 BROWN1) without loss of meaning. This, then, is the second consolidation rule: if a triple of the form (A O B) precedes a triple of the form (O B C), merge the two triples into the single triple with format (A B C).

MUSE also uses a consolidation routine to solve the problem of how to account for such sentences as "chase be pursue down street.". The triple notation for such a sentence is (CHASE1 BE1 PURSUE1) (PURSUE1 DOWN1 STREET1) which reduces to (CHASE1 O PURSUE1) (PURSUE1 DOWN1 STREET1) after the application of B-actions. CHASE1 has superset PURSUE1, but the prepositional phrase DOWN1 STREET1 should modify CHASE1, not PURSUE1. The reason for this lies in Quillian's definition of a concept given on page 462 of his 1969 paper: "All properties pointed to in a unit represent predicates which, when associated^{*} with that

* a property is associated with the superset, but connected to the unit.



unit's superset, constitute the concept the unit represents." In this example DOWN1 STREET1 is the property, CHASE1 is the unit, and PURSUE1 is the superset which by Quillian's definition implies that DOWN1 STREET1 should be connected to CHASE1 not the superset PURSUE1. Based on this example is the third consolidation rule: if a triple of the form (A O B) is discovered, interchange A and B in this triple; and in all other three unit triples which follow this triple replace A by B and B by A wherever they occur. Thus, the new notation is (PURSUE O CHASE) (CHASE1 DOWN1 STREET1). There are three things of importance about this rule. First, no triples which precede the (A O B) triple are affected, so that sentences such as "chase down street be pursue." will be interpreted correctly. Second, any two unit triples which follow will not be reversed, thus allowing more than one superset to be defined as in the sentence "ruff be dog that be animal." which reduces to (RUFF1 O DOG1) (DOG1 O ANIMAL1) before consolidation and (DOG1 O RUFF1) (ANIMAL1 O DOG1) after consolidation. Third, triples defining supersets such as (A O B) are now reversed, that is the first element is now the superset of the third (e.g. (DOG1 O RUFF1) rather than (RUFF1 O DOG1)). In the "ruff chase volkswagen." example, no B-actions are performed, and no consolidation is required.

Semantic Processing of the Input

So far in its processing, MUSE has reduced the current parse tree to a set of relational triples which have been acted upon and consolidated to as compact a form as possible. The model now tries to discover if the triples are semantically well-formed. MUSE assumes that each triple is related to a memory concept, and it proceeds to try to discover exactly what concept, if any. This is accomplished by using the fact that the triple notation is very similar to the notation of the semantic memory, that is the two unit triples can be compared to subset-superset connections in memory, and the three unit triples are very similar to memory units modified by properties.

MUSE looks at each triple in turn, starting at the left-most triple. If the triple being examined is a two unit triple, it has the form (A O C) where A and C are memory units and O is the null unit. Since two unit triples represent subset-superset relations in memory, MUSE assumes that unit A lies somewhere on the superset chain of unit C. The model iterates the superset chain of C until it finds A, or until it arrives at the end of the chain. If MUSE does not find A, the triple is judged to be "semantically ill-formed" (opposite of semantically

well-formed), and is flagged accordingly. If A lies on the superset chain of C, however, the triple agrees with the semantic memory and is regarded as semantically well-formed. In either case, MUSE proceeds to the next triple.

If the triple currently under consideration is a three unit triple (i.e. of the form (A B C)), it is assumed to represent a unit-property relationship in memory (i.e. the unit being A and the property (B C), often indicated by the notation A-(B C)). MUSE attempts to find a semantically related unit-property combination in the memory. Before going further, recall the definitions of semantic identity and superset intersection in memory, remembering especially that two units can be semantically identified if they have a superset intersection in memory. The notion of semantically related unit-property can now be defined. Assume that there is a three unit triple of the form (A B C), and that there is a unit D in memory having a property with attribute E and value F. Then the unit-property D-(E F) is said to be semantically related to the three unit triple (A B C) if D can be semantically identified with A, E can be semantically identified with B, and F can be semantically identified with C, and if D does not contain any properties which "contradict" (B C). Two properties

(B C) and (G H) are contradictory if (i) B and G are the same unit; (ii) C and H are different units; and (iii) C and H are not connected via a superset chain. For example (COLOR RED) and (COLOR BROWN) are contradictory properties. For convenience, the superset chains used in finding semantic relationship are assumed to be at most three units long. If there is a unit-property in memory which is semantically related to the current triple, the triple is said to be semantically well-formed. If there is no unit-property so related, the triple is semantically ill-formed, in which case it is flagged. MUSE goes on to the next triple regardless of semantic well-formedness.

After MUSE has processed every triple in the current set of triples, it checks for any flags indicating the existence of semantically ill-formed triples in the set. If there are one or more semantically ill-formed triples, the current candidate combination is considered to be unrelated to the memory and is rejected, at least for the present, as a possible meaning for the input. However, the candidate combination could possibly still be valid, especially if some new concept has been defined in input, so the model places the current triple set on a stack of called the "failed triple" stack. This stack is arranged in order of most likely sets of triples to least likely

sets of triples and is truncated at three such sets, the less likely dropping off for the more likely. One set of triples is said to be "more likely" than another set if it contains fewer flagged triples; if the number of flagged triples is the same, the set which was defined first is considered to be more likely. After stacking the invalid triple set, MUSE goes back for the next candidate combination.

The "ruff chase volkswagen." example has yielded one triple: (RUFF2 CHASE1 VOLKSWAGEN1). MUSE now tries to find out if there are any memory unit-properties which are semantically related to this triple. Since it is unlikely that there is a unit-property even remotely related to "a band around the neck" chasing a volkswagen, the model will likely find no semantically related unit-property and will reject (RUFF2 CHASE1 VOLKSWAGEN1) as being semantically ill-formed. Thus, syntax has led to the rejection of the candidate combination

RUFF3,CHASE1,VOLKSWAGEN1; and semantics has led to the rejection of the combination RUFF2,CHASE1,VOLKSWAGEN1. The triple (RUFF2 CHASE1 VOLKSWAGEN1) is now added to the failed triple stack, and MUSE returns to process the only remaining candidate combination RUFF1,CHASE1,VOLKSWAGEN1. Eventually a triple (RUFF1 CHASE1 VOLKSWAGEN1) results, which is tested for semantic well-formedness. Recall

that the concept of Figure 2.1 is stored in the semantic memory, that is the memory contains a unit RUFF1 which has the property (CHASE1 CAR1). It is clear that the memory unit RUFF1 can be semantically identified with the triple unit RUFF1, being the same unit. Also the memory unit RUFF1 has a property whose attribute CHASE1 is the same as the second triple unit, and whose value CAR1 is on the superset chain of the third triple unit VOLKSWAGEN1. Since there are no contradictions, the memory concept shown in Figure 2.1 is semantically related to the triple (RUFF1 CHASE1 VOLKSWAGEN1), and, hence, the triple is semantically well-formed.

Output from MUSE

Once the model has discovered a set of semantically well-formed triples, it assumes that it has discovered an interpretation for the input, and it prints "THE INPUT CAN BE INTERPRETED AS FOLLOWS:". The English version of the interpretation is now printed out, accomplished by replacing each unit in each triple with its associated dictionary entry. On each output line is printed a triple followed in brackets by the unit-property that is semantically related to the triple. This gives the monitor a better idea of how MUSE arrived at its interpretation. To further help the monitor,

each word of the output has appended to it a number to show what meaning of the word has been chosen (such as RUFF1, RUFF2, or RUFF3). Of course, the output routine must take special care to indicate the subset-superset relationship defined by two unit triples. It does this by inserting the words BE and SUPERSET-OF in the English output. Thus, for example, if the two unit triple (DOG1 O RUFF1) is given to the output routine it prints "RUFF1 BE DOG1 (DOG1 SUPERSET-OF RUFF1)." The "ruff chase volkswagen." example would have output "THE INPUT CAN BE INTERPRETED AS FOLLOWS:" followed on the next line by "RUFF1 CHASE1 VOLKSWAGEN1 (RUFF1 CHASE1 CAR1).".

It should be mentioned at this point that if the model finds no semantically well-formed interpretations for the input, it unstacks the most likely triple set from the failed triple stack, and assumes that this is an interpretation for the input. This set is printed out as above, triple by triple, with the difference being that no unit-properties have been related to the triples meaning that the brackets behind the triples (except for two unit triples which define subset-superset relationships) are left empty. MUSE indicates its uncertainty about the interpretation by preceding the out with "I AM NOT SURE BUT I THINK THAT" and on the next line

"THE INPUT CAN BE INTERPRETED AS FOLLOWS:". This is then followed by the appropriate triples and empty brackets; for example "RUFF2 CHASE1 VOLKSWAGEN1 ().". From this point on, the processing of an interpretation from the stack is the same as the processing of an ordinary interpretation.

After an interpretation has been printed out, MUSE must confirm that it is valid. Thus, it asks "IS THIS A VALID INTERPRETATION?" and waits for a "yes" or "no" answer from the monitor (actually the model tests for "no" only, so any response not containing "no" is considered to be affirmative). If the answer is "no", the model goes back and tries to derive a new interpretation, either by processing new candidate combinations or, if none of these are available, by unstacking more triple sets from the stack. When the model has exhausted even the failed triple stack, it assumes that no interpretation is possible and types "I CANNOT INTERPRET THE INPUT." and then "I AM PROCEEDING TO THE NEXT STATEMENT." before returning to get a new input sentence.

However, if at any stage a valid interpretation is found, MUSE asks the monitor "DO YOU WANT THIS ADDED TO MEMORY?". The monitor once again responds with either a "yes" or "no". If the answer is "no", the model goes

on to the "C-action" portion of the program, making sure that the memory files are unchanged from their state before the sentence was entered. If the answer is not "no", MUSE adds the new interpretation to memory before proceeding to the C-actions.

Adding an interpretation to memory is not a difficult task, since the triple notation is so compatible with the memory notation. MUSE once again proceeds from left to right, processing each triple in turn. If the triple is a two unit triple, say (A O C), A is added to C as the superset of C, unless C already has a non-null superset, in which case it is left alone. Hence, new subset-superset relationships can be defined in memory. If the triple is a three unit triple such as (A B C), the model adds the property with attribute B and value C to the property file. A pointer to (B C) is then attached either to a memory unit or to another property, which alternative depending on the position of A in the first triple in which it has appeared (i.e. A might have occurred in an earlier triple than (A B C)). If A first appeared as the first or third element of a triple, then A represents a unit in the memory notation and (B C) is pointed to as a new property of the unit A. However, if A first appeared as the second element of a triple, say (E A D), then it has already been defined as the attribute of the property (A D) which implies that

(B C) is the subproperty of a property, not of a unit. Hence, a pointer to the property (B C) is added to the property (A D). In this manner complicated subproperties can be built up by MUSE as it processes each triple in the relational triple notation. Of course, checks are made so that duplicate property pointers are never added to a unit or property. Also, if MUSE has discovered a new property which is identical (same attribute, value, and subproperties) to a property already in memory, the new property is not added to the property file, and all pointers to the new property are changed so they point to the property already defined.

For example, if the relational triple notation consists of the three triples (ROVER1 CHASE1 VOLKSWAGEN2) (CHASE1 DOWN1 STREET1) (VOLKSWAGEN2 COLOR1 GREEN1), the following procedure is employed to add the concept to memory. After the property (CHASE1 VOLKSWAGEN2) has been added to the property file, a pointer to the property is appended to the unit ROVER1, since ROVER1 first occurs in the first element of a triple. Next, (DOWN1 STREET1) is added to the property file. (DOWN1 STREET1) is also attached as a further subproperty of (CHASE1 VOLKSWAGEN1), since CHASE1 first appeared as the second element of a triple. Finally, the property (COLOR1 GREEN1), after being added to the property file, is pointed to from the

unit VOLKSWAGEN2, since VOLKSWAGEN2 was first seen as the third element of a triple. In the "ruff chase volkswagen." example, recall that the triple notation is (RUFF1 CHASE1 VOLKSWAGEN1). MUSE would add the new property (CHASE1 VOLKSWAGEN1) to the property file and would then set up a pointer to the property from the unit RUFF1. Thus, RUFF1 now has now properties: (CHASE1 VOLKSWAGEN1) and (CHASE1 CAR1).

Motor Actions in MUSE

MUSE has now arrived at the "C-action" portion of its processing. The C-actions are the motor actions that perform "useful" tasks such as halting etc. Generally these actions are associated with units that can be used as imperative verbs, although the "equivalent" and "?" actions are not of that variety. Before looking for the existence of C-actions in the interpretation, however, MUSE first asks "DO YOU WISH ME TO ACT, IF POSSIBLE?". If the answer is "no", the model does not attempt any C-actions and returns for a new input sentence. If the answer is not "no", the system looks for C-actions.

MUSE first looks to see if "?" is the punctuation at the end of the input sentence. If it is, the model assumes that the interpretation which it has given the

input contains all the information required to answer the question which has been posed, and points out "THE ANSWER TO YOUR QUESTION WAS GIVEN ABOVE." before proceeding to the next input. Notice the implications of this action: that MUSE contains no explicit question answering routines; it merely uses the interpretation processes already defined. Thus, MUSE can answer "who", "what", "where", and "when" questions only if the appropriate answer is a subset of the "wh" word. For example, if the system knows that "john have volkswagen.", and if in the semantic memory JOHN1 has superset PERSON1, and PERSON1 has superset WH01, then the system can correctly answer the question "who have volkswagen?" by substituting JOHN1 as a candidate unit for "who". Similarly a "yes-no" question such as "john have volkswagen?" will not be answered yes or no. Instead, it will be answered by the interpretation "JOHN1 HAVE1 VOLKSWAGEN1" if the sentence is true; and will be answered by "I CANNOT INTERPRET THE INPUT." if the sentence is false. The C-action associated with the "?" only confirms that a question has been asked - it is up to the monitor to find the answer from the interpretation.

If MUSE has found no "?" action, it takes each triple (A B C) of the interpretation and looks for the occurrence of a C-action linked with the B unit of the

triple. If the system finds such an action, it checks to see if the action is the "equivalent" action. If so, the model prints "EQUIVALENCE DEFINED." and transfers control to the appropriate routines. The "equivalent" action (at present associated only with the unit representing the English word "equivalent") assumes that unit A and unit C of the triple are to be amalgamated into one unit in memory; that is the model removes unit A and consolidates it into unit C. The action first changes the dictionary so that the word associated with unit A will no longer point to unit A, but instead to unit C. All properties pointed to from unit A that are not already defined under unit C are then copied to C. The action then removes the unit A from the semantic memory and changes any pointers to unit A to pointers to unit C. This action is used to amalgamate different English words which mean the same thing, such as, for example "ass" and "donkey". When the "equivalent" action is finished, MUSE asks for new input.

If no "equivalent" action is found, MUSE looks for the so-called "imperative" actions which command the model to do things. There are six of these actions: "halt", "quit", "define", "describe", "change", and "update", each associated with the indicated verb. The model will only transfer control to the appropriate action if (i) the action is linked to the middle unit

of the triple and (ii) if the first unit of the triple is SELF1 or has SELF1 on its superset chain. Hence, the triple (SELF1 HALT1 PROCESSING1) will be acted upon, whereas (SELF1 EAT1 CAKE1) or (PERSON1 HALT1 PROCESSING1) will not be acted upon. The six "imperative" C-actions will now be explained.

The "halt" action performs the same processing as the "stop and store" fast exit ":" described earlier in the chapter. The action stores all the memory files, prints "THANKS FOR THE CONVERSATION." and then stops processing. The "quit" action is the same as the "+" fast exit; that is, the model prints "THANKS FOR THE CONVERSATION." and stops immediately with no storing of memory files. Neither "halt" nor "quit" pay any attention to what the C unit in the triple is, so that "halt cake." or "muse quit crying." would still result in the action being performed.

The "define" action is similar to the "?" action in that it is in reality a dummy action. The action simply types "THE CONCEPT HAS BEEN DEFINED." and proceeds to the next input. This is because MUSE has already added any new concepts to memory as part of its comprehension process and, thus, needs to perform no further processing. The "define" action is usually used when

some syntactically correct way is needed to add a single word to memory. For example, if the monitor wanted to add the word "dog" to memory, he could not just type "dog (n;)." because the input would be ungrammatical; but he could type "define dog (n;)." and have the system understand him, since the sentence is syntactically correct.

The "describe" action is perhaps the most interesting of all the C-actions. It is used when the monitor wants to see an English version of the memory's concept of some word. The action looks up the C-unit of the (A B C) triple in the file of units, collecting its superset and all its subproperties. These are then translated into English and the unit and its superset are printed out, followed by each property separated by commas. If the unit has no superset or no properties, these are not printed in the output. Thus, the concept for RUFF1 (which now has the new property (CHASE1 VOLKSWAGEN1) as well as the old (CHASE1 CAR1)) could be printed out by typing "describe ruff." to which MUSE would respond "RUFF1 BE A UNIT IN MEMORY." followed on the next line by "RUFF1 BE DOG1: CHASE1 VOLKSWAGEN1, CHASE1 CAR1,.". After performing the action, the model returns for a new input statement.

The final two actions "change" and "update" help the monitor to perform various editing tasks on the memory files. These actions key on the C unit of the triple, which specifies which file is to be "changed" or "updated", but if no specific file is designated, the model automatically starts at the dictionary file.

If the action is "change", the model attempts to edit the particular file specified. The system asks the monitor to specify what word, unit, or property he wants to change (depending, of course, on what file is being acted upon). The model then types out the old definition of the particular item being changed, and asks the monitor to type in a new version. After this has been done, the model asks if there are any more items to be changed from the current file. The monitor responds either with another item or, if he wants to quit, with a ":". The ":" indicates to the model that the current file has been changed sufficiently, so the model proceeds to the next file (in order dictionary file, unit file, property file) and asks if that file is to be changed. This process continues until the changing of the property file is finished, whereupon the model asks the monitor "DO YOU WANT TO CONTINUE CHANGE?". If the monitor answers "no", the model goes back for a new input sentence; but if he does not respond with a

"no", the "change" action is repeated all over again, starting with the dictionary file. Note that the syntax file cannot be changed (any attempt to do so results in a message from MUSE), due to the way it is defined (see Chapter IV).

The "update" action adds new words, units, properties, and parse trees to their respective files. It works in a manner similar to "change", cycling through the files starting with the specified file and continuing until the syntax file has been updated. When processing a file, the model prompts the monitor for each part of the item to be updated, and once again, a ":" response tells the model to proceed to the next file. After the syntax file has been acted upon, the model queries "DO YOU WANT TO CONTINUE UPDATE?". As before, a "no" answer sends MUSE for new input, and a non-negative response starts the update again with the dictionary file.

If MUSE can find no actions, as in the (RUFF1 CHASE1 VOLKSWAGEN1) triple, it types "NO ACTION POSSIBLE." and returns for a new input sentence. More details of MUSE and a sample run will be given in Chapter IV where the implementation of the model is discussed.

CHAPTER IV

IMPLEMENTATION OF MUSE

Introduction

MUSE is currently implemented as 1172 line SNOBOL4 program running interactively under Michigan Terminal System (MTS) on an IBM 360/67 computer. The details of the program will not be given here, but a sample run will be examined in depth. Before this run can be understood, however, much needs to be explained about the data files which MUSE uses, the initializations that the program requires, and the machine response format which has been defined.

Data Files in MUSE

MUSE uses four files containing its dictionary, units, properties, and parse trees (see Appendix I for examples). These files, each defined as a separate line file under MTS, have been called respectively DICTION, UFILE, PFILE, and SYNTAX and have many similarities. At the front of every file is an integer stating the number of items in the file. Items (i.e. words, units, properties, or parse trees) are separated by periods; and each file terminates with a colon. These files can be updated or changed using MTS'

editing features, or they can be revised using procedures in MUSE itself. Any new words, units, properties, or parse trees that are defined using the procedures of the program are added to the front of these files, so that when MUSE is interpreting future input, the most recent items will be tried first.

DICTION is the file which contains the dictionary for MUSE. In this dictionary are all the English words which are defined in the system, plus pointers from each word to corresponding units in UFILE. A typical entry in the dictionary might be .FAST; U22, U21,. with periods delineating the beginning and end of the dictionary entry. The word "fast" has two meanings in MUSE, one represented by the unit U22 and the other by the unit U21. For convenience these are called FAST2 and FAST1 respectively, where a larger numeric suffix indicates more recent definition.

UFILE, one of two files which make up the semantic memory for MUSE, contains all the units which are pointed to from DICTION. A typical unit is U22, the FAST2 unit, whose UFILE entry is .U22; B; A3;; U23; PO,. . . The first element, U22, is the name of the unit. The second element, in this case B, is the part of speech symbol encoded as follows: N for noun, P for pronoun, V for verb, E for linking verb, B for adverb, D for adjective, R for article,

T for preposition, I for interjection, C for relative pronoun, and J for co-ordinate conjunction. The next element of the unit has the actions which are associated with the unit, in this case A3. A1, A2, and A3 stand for the first, second, and third A-actions; B1 and B2 are the first and second B-actions; and C1 is halt, C2 is update, C3 is equivalent, C4 is the ? action, C5 is define, C6 is describe, C7 is quit, and C8 is change. Only one action from each class (A, B, and C) is allowed per unit; and if a unit has no actions, the space is left empty. U23 in the example is the pointer to the unit's superset, which happens to be the unit SPEED1. If a unit has no superset, it is assigned the null superset U0. If a new unit has been defined on-line by MUSE, a superset called UA is temporarily assigned to the unit to mark the unit as new. After the input sentence containing the new unit has been processed, the UA is replaced by U0 or by a new superset derived from the input. The final section of the unit contains pointers to modifying subproperties which are arranged in order of recency and are stored in PFILE. In the present unit the property is P0, the null property, defined for all units with no modifying properties.

The other half of the semantic memory is contained in PFILE. PFILE contains properties looking much like

.P6; U11; U1; PO,. , where P6 is the code indicating that this is the sixth property and U11 and U1 are pointers to units representing the attribute and value of the property. After the attribute and value come pointers to any further subproperties, and if there are none, as in the current example, the null property PO is used.

The syntactic component of MUSE is stored in the SYNTAX file in entries which contain the parse trees for various sentence types. Such an entry might be .(2)VE, NPVEBDRTICJ, +S1; ZO, V1 - V1; Z1, Z2-. which will parse a sentence such as "halt processing.". The brackets contain the number of words in the sentence type, in this sentence 2. Immediately following are the parts of speech (separated by commas) which are valid for each word of the sentence type. In the example the first word of the sentence can be a verb or a linking verb, and the second word can be any part of speech. By allowing more than one part of speech for every word, more generality and flexibility is attained.

The third element of the SYNTAX entry, starting at the plus sign and continuing until the period, is the parse tree for the sentence type. The parse tree here is S1; ZO, V1 - V1; Z1, Z2-. The tree is described from the top down, each element (such as S1, ZO, V1, etc.) representing one node of the tree. An element preceding

a semi-colon (such as S1) is the head of a branch pair, and the two elements following the semi-colon but preceding the minus sign (such as Z0, V1) are the left and right branches from that head. Each branch pair is described in this fashion until the entire tree has been coded. The letter in an element (i.e. S, Z, V, etc.) identifies the node type as one of the following: sentence or clause (S), noun phrase (N), verb phrase (V), adverb phrase (B), adjective phrase (D), prepositional phrase (T), and terminal node (Z). The numbers after the elements distinguish that node from others of the same type (e.g. two noun phrases are N1 and N2). The numbers after terminal nodes have special significance, however, because they represent corresponding words of the sentence type (i.e. Z1 is the first word of the sentence type, Z2 the second, and so on). Z0 is the null node to be filled in later by actions such as A2 and A3.

The parse tree S1; Z0, V1 - V1; Z1, Z2- , if drawn as a tree in memory notation, would look like Figure 4.1. If the

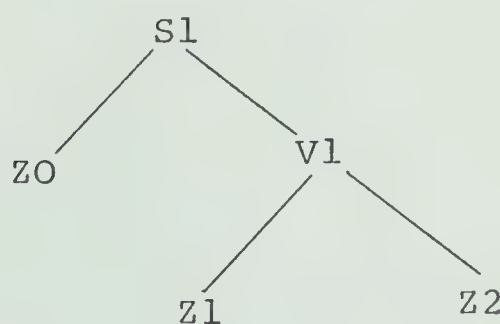


Figure 4.1 Parse tree in SYNTAX notation

input sentence is assumed to be "halt processing.", the parse tree could be translated into English terms as in Figure 4.2.

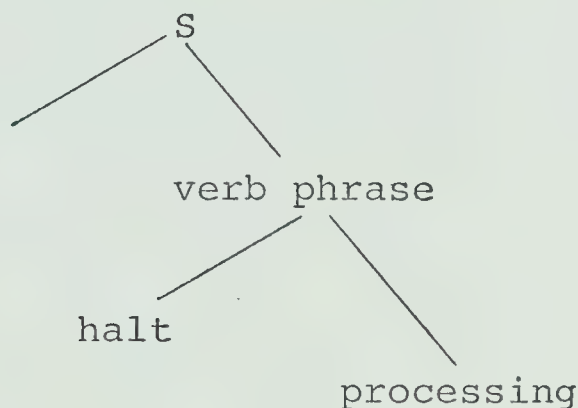


Figure 4.2 Parse tree in English

In Appendix I there are two sets of memory files, the first set containing MUSE's "knowledge of the world" before the sample run has taken place, and the second set showing the knowledge which MUSE has gained during the run. The first set of files contains several points of interest. DICTION contains 31 different words, two of which (OF and FAST) have multiple meanings. Note that each action word (such as HALT, DESCRIBE, etc.) has been defined, that the file names (DICTION, UFILE, etc.) are words in memory, and that the model has a concept of SELF and knows its own name MUSE.

UFILE contains 33 units representing concepts that the system understands. There are two more units in UFILE than there are words in DICTION due to the extra meanings

for OF and FAST. Both OF and FAST have two meanings in memory represented as OF2, OF1, FAST2, and FAST1.

OF2 (U30) is the meaning of "of" which could occur in a phrase such as "color of volkswagen" that reduces to triple notation (VOLKSWAGEN2 O COLOR1). The other unit for OF, OF1 (U28), could be used in "volkswagen of john" meaning "john have volkswagen". The two meanings of FAST, FAST2 (U22) and FAST1 (U21), are respectively the adverb used in the sense "speed fast", and noun meaning "going without food". An interesting feature of UFILE is the fact that the unit PERSON1 (U15) has superset WHO1 (U5), implying that question answering can be undertaken with PERSON1 as a candidate unit for WHO1. The units MUSE1 and YOU1 (U3 and U2) both have superset SELF1 (U1), indicating that they can be used as subjects of sentences which "command" the model.

The unit SELF1 has pointers to six properties, the only properties defined in the model at this point. Contained in PFILE, these properties are (DESCRIBE1 SELF1), (CHANGE1 THING1), (QUIT1 THING1), (DEFINE1 THING1), (UPDATE1 THING1), and (HALT1 THING1). Note that whenever the model uses properties that apply to THING1, it can deduce that they apply to the file names, as well, since DICTION1 (U19), UFILE1 (U17), PFILE1 (U18), and SYNTAX1 (U16) have superset THING1 (U14).

The SYNTAX file contains 14 parse trees for 14 sentence types that are from two to eight words in length. Because there must be exactly two branches for all non-terminal nodes in a parse tree, articles cannot be coded as extra third branches as is usually done. Thus, dummy noun phrases are devised to contain the article, and these are later deleted from the tree by action A1. Null branches (Z0) are quite common, are usually associated with adjective phrases, adverb phrases, and imperative sentences, and are usually filled in during processing with the help of actions A2 or A3. Also noteworthy is the first sentence (of six words), the only sentence which contains a subordinate clause (delineated in the parse tree by S2). Many part of speech vectors with multiple entries have been specifically inserted to allow the definition of new words. For example, a two word sentence has been designed for such sentences as "define dog.", where the second word can be any part of speech. Finally, several syntactic ambiguities are apparent, such as the two parse trees for the eight word sentence.

The Sample Run : Memory Build Up

The sample run is divided into a memory build up section and a question answering section, each presented in Appendix II. There are 26 sentences and questions

which are handled by the model, to which numbers have been added for convenient identification. Input by the monitor is in small letters, output from MUSE is capitalized, and all words taken from the semantic memory have numbers affixed to them so that different word meanings can be distinguished. Modifications and additions to the memory during the sample run can be best understood by comparing the first and second sets of files in Appendix I.

MUSE is started by typing the MTS "\$RUN" command shown at the beginning of Appendix II. The \$RUN command identifies relevant files and input-output devices and initiates program compilation. A successful compilation results in the printing of the statement "O SYNTACTIC ERROR(S) IN SOURCE PROGRAM" after which control is transferred to MUSE. The model types "GOOD MORNING. MY NAME IS MUSE." followed by "PLEASE TYPE YOUR INPUT.". It then waits for the monitor to enter an input statement.

The first sentence to be processed is "update syntax.". MUSE interprets this input by adding the imperative subject SELF1 to the sentence (action A2), after which the input can be semantically related to the memory unit-property SELF1 - (UPDATE1 THING1). Since this is an interpretation for the input, MUSE

prints out both the current triple notation of the sentence and its related unit-property in brackets. The monitor affirms that the interpretation is valid, that it need not be added to memory, and that it should be acted upon. The update action is then embarked upon, starting at the file SYNTAX since this was specified in the input. The monitor adds a new parse tree (for sentences such as "chase means pursue down street.") after which a colon is entered indicating that no more SYNTAX update is needed. MUSE, after adding appropriate brackets, plus signs, and periods to the new SYNTAX entry, tacks the entry to the front of the SYNTAX file and increments the number of parse trees to 15. Since the update action finishes after SYNTAX update, the model asks the monitor if he wants to continue updating; and since he does, MUSE proceeds to DICTION update. The monitor enters the new word "means", which MUSE adds to the front of DICTION and assigns the unit pointer U34. Since there are no more words to be added to the dictionary, the model carries on to the next update, the UFILE update, where the monitor enters the part of speech "e", the action "bl", the superset "u0", and the subproperty "PO,". MUSE then sets up the new unit U34 (at the front of UFILE) to which these elements are added. Thus, a new word

"meens" with corresponding unit U34 has been set up in memory. Next, the monitor constructs a new property (with attribute U10, value U14, and no subproperties) which MUSE designates as P7 and adds to the beginning of PFILE. Since there are no new SYNTAX items, and since the monitor does not want to continue, the update action is terminated, and the model asks for new input.

Notice that the monitor has misspelled "means", has added the property (U10 U14) (already defined as P3) instead of (U11 U14), and has not added the new property as a property of the first unit SELF1. Thus, he now types "muse change diction." and makes changes to the appropriate files. The model can interpret the input since it knows SELF1 - (CHANGE1 THING1); and MUSE1 is a subset of SELF1 and DICTION1 is a subset of THING1. Once the change action begins, the model asks the monitor to change each file except SYNTAX (since SYNTAX items have no unique labelling system as do words, units, and properties). When changing an element of the file, the monitor enters the word, unit, or property to be changed, the model responds with the old version, and the monitor enters a new version to replace the old one. In this manner "meens" becomes "means", the property P7 is added to U1, and P7 is changed to (U11 U14) before the model proceeds to the next input.

For the next sentence, "person change pfile.", the model cannot find a semantically related unit-property in memory, so it goes to the failed triple stack for the closest approximation to the meaning of the input. Since this is a valid interpretation and is not added to the memory, MUSE asks if it is to act. The monitor says "yes", but no action is possible since MUSE cannot recognize the subject as referring to SELF1 or any subset of SELF1.

The next two input sentences (numbers 4 and 5) add new concepts to the memory. Sentence number 4, "a(r;al,) equivalent the(r;al,).", shows that new words can be defined in text and that the equivalent action works. What is contained inside the brackets of the two new words is of the same format as the UFILE entries; in fact, the bracketed elements are inserted directly into the new units. Hence, "a" and "the" are both articles with the first A-action. The importance of suppressing the actions the first time through is illustrated here, since without this suppression, action A1 would have deleted both new units from the input before it could have been interpreted by MUSE. Once the input has been understood by the model, it is added to memory; that is, (EQUIVALENT1 THE1) is added as a property of the unit A1. Finally, equivalence is defined between A1 and

THE1 yielding one unit, THE1, with one property, (EQUIVALENT1 THE1), which has been transferred to THE1 from A1.

Sentence 5 shows the definition of a unit, a superset, and a subproperty all in the statement "chase(v;) means pursue(v;) down(t;) street(n;).". It also illustrates the reason for the definition of the third consolidation rule of the last chapter. Since "means" has been defined by the update and change actions to be the same as "be", this sentence is interpreted (through the use of the failed triple stack and the third consolidation rule) in the following manner: CHASE1 has superset PURSUE1 and subproperty (DOWN1 STREET1). The property (DOWN1 STREET1) when associated with the superset PURSUE1 gives the memory's concept of CHASE1, a valid definition satisfying Quillian's criterion of a memory concept quoted in Chapter III (also in Quillian, 1969, page 462). The new units, new superset, and new property are all added to the memory, and no further actions are possible.

Sentence 6, "define dog.", in addition to demonstrating the define action, starts off a small story about dogs and cars which the model is fed sentence by sentence. It is from information learned in this story that MUSE answers its questions in the second part of the sample

run. The next input "ruff(n;) be dog(n;) that be animal(n;)." informs the system of two levels of superset in the same sentence, as well as defining three new memory units. There are two interesting points about this statement: (i) the deletion of the word "that" by the model when interpreting the input (which shows the way it handles clauses), and (ii) the fact that the second consolidation rule (amalgamating two unit triples) is sophisticated enough not to join two unit triples of the form (A O B)(B O C) which define subset-superset relationships.

Input sentence 8, "volkswagen(n;) be car(n;).", defines subset-superset relation between two new units, VOLKSWAGEN1 and CAR1. Sentence 9 is "volkswagen be red.". It has exactly the same format as sentence 8, and could be parsed using the same parse tree that sentence 8 has used. But this would define RED1 as the superset of VOLKSWAGEN1, an obviously false interpretation. Hence, MUSE rejects the sentence 8 parse tree and uses another tree especially designed to handle the unit-property concept of sentences such as sentence 9. MUSE has thus deciphered a syntactic ambiguity. This example also marks the first application of action A3 where the superset of RED1 (namely COLOR1) is added to the tree.

Of interest, as well, is the use of the second consolidation rule to amalgamate the two unit triples (VOLKSWAGEN1 O COLOR1) and (O COLOR1 RED1).

Input sentence 10 ("ruff chase the red volkswagen down the street.") is the longest sentence that can be understood using the current SYNTAX file.

There are two parse trees for the sentence, the first where the preposition phrase "down street" modifies the noun "volkswagen" (i.e. it is the volkswagen which is down the street), and the second which has "down street" modifying the verb "chase" (i.e. the car is chased down the street). MUSE starts by applying the first parse tree to the sentence. This attempt is rejected in the semantic processing portion of the model and its triple notation (RUFF1 CHASE1 VOLKSWAGEN1) (VOLKSWAGEN1 COLOR1 RED1) (VOLKSWAGEN1 DOWN1 STREET1) is stacked at the top of the failed triple stack. The second parse tree is now tried and results in the triple notation (RUFF1 CHASE1 VOLKSWAGEN1) (CHASE1 DOWN1 STREET1) (VOLKSWAGEN1 COLOR1 RED1) which is also rejected by MUSE's semantic intersection techniques. This triple set is then added to the failed triple stack, but above the first attempt since two triples (CHASE1 DOWN1 STREET1) and (VOLKSWAGEN1 COLOR1 RED1) are semantically related to memory, whereas in the first try only the triple

(VOLKSWAGEN1 COLOR1 RED1) matches. There are no more possible interpretations, so MUSE prints out the first triple set on the stack, that is, the set associated with the second parse tree. This is a correct interpretation, which means MUSE has resolved a syntactic ambiguity by using semantics.

Sentence 11, "ruff chase the volkswagen fast.", shows how syntactic processing can resolve a semantic ambiguity. The word "fast" means both "going without food" (FAST1) and "speed fast" (FAST2), and without syntax MUSE has no way of telling which meaning is appropriate here. However, by using the SYNTAX file, the model finds only one valid parse tree for the sentence, the tree applying to the adverb FAST2, and hence the ambiguity disappears.

MUSE has no trouble with the next sentence, "john(n;) be person.", which merely defines "john" as being a person (and hence a candidate for "who"). Sentence 13, "john have a green volkswagen(n;).", defines a green volkswagen (added as VOLKSWAGEN2) owned by John. Thus, MUSE has added another possible ambiguity since it now knows of two volkswagens, which can only be told apart by their colors. The next input is "this volkswagen be a car also." which has several interesting idiosyncrasies. The sentence contains no

less than three words ("this", "a", and "also") that are inserted just to improve readability and are deleted early in the processing by action A1. This is also the first sentence of the sample run which the model has misinterpreted, a result of the fact that MUSE already knows that VOLKSWAGEN1 has superset CAR1, but it does not know this fact about VOLKSWAGEN2. Thus, after first trying to match VOLKSWAGEN2 and CAR1 in memory and failing, MUSE stacks the interpretation and finds the connection between VOLKSWAGEN1 (the other candidate unit for "volkswagen") and CAR1 which it prints out. Since this is not a valid interpretation, the first try is unstacked and printed out, yielding the correct comprehension of the input.

The model now processes sentence 15, "color of the volkswagen be green.", a paraphrase of a fact that it has already learned in sentence 13. The model must choose between two meanings of the word "of", OF2 and OF1, and also two meanings of "volkswagen", VOLKSWAGEN2 and VOLKSWAGEN1. It cannot tell which meaning is correct for each word from syntactic considerations, since both candidates in each case have the same part of speech, so MUSE must use the context of the sentence to solve the ambiguity (i.e. semantic processing to resolve a semantic ambiguity). MUSE assumes that OF2 and VOLKSWAGEN2

are the candidate units for "of" and "volkswagen" respectively (since they are the most recently defined units), then deletes the word "the" from the sentence, adds the superset COLOR1 to the parse tree, and breaks the sentence into triples (COLOR1 OF2 VOLKSWAGEN2) (COLOR1 BE1 COLOR1) (BE1 COLOR1 GREEN1). Application of action B2 on OF2 and B1 on BE1 yields the triple notation (VOLKSWAGEN2 O COLOR1) (COLOR1 O COLOR1) (O COLOR1 GREEN1), which is consolidated by the first consolidation rule into (VOLKSWAGEN2 O COLOR1) (O COLOR1 GREEN1) and by the second consolidation rule into (VOLKSWAGEN2 COLOR1 GREEN1). This triple is then compared with the memory concept already known from sentence 13, is found to be semantically related, and is printed out. Thus, sentence 15 has not only shown how MUSE can use semantic processing to resolve semantic ambiguities, but also how the model employs its consolidation rules and its A and B actions to understand paraphrases. Note that this is the first sentence where the monitor does not want any actions to be attempted, so the model proceeds straight to the next input.

Sentence 16 merely teaches MUSE that there is another dog called "rover" (i.e. "rover(n;) be another dog."), but sentence 17 is perhaps the most interesting

and surprising sentence of the entire sample run.

"rover chase the green volkswagen." states a brand new concept which MUSE has never encountered before (that ROVER1 chases VOLKSWAGEN2), and it seems reasonable to assume that the input will be uninterpretable except by taking close approximations from the failed triple stack. But it turns out that MUSE can understand the input directly from concepts which it already knows without appealing to the failed triple stack. Using the fact that another dog, RUFF1, chases another car, VOLKSWAGEN1, the model can infer that (ROVER1 CHASE1 VOLKSWAGEN2) is semantically well-formed. This implies that the model has, in effect, generalized the concept "dogs chase cars" from the specific fact that "ruff chases volkswagen.". In other words, the model can learn by example.

Sentence 18 is a sentence which is uninterpretable since there is no parse tree for "john chase dog be" in SYNTAX. Sentence 19, "halt processing.", tells MUSE to stop and store its memory files (action C1), thus saving the information which the model has learned in the last eighteen sentences. This sentence is followed by a sample of the MTS termination format which shows all of the statistics for the run. The average execution time per sentence can be derived from these statistics and turns out to be a fairly slow 50 seconds/sentence.

The Sample Run : Question Answering

Question answering is carried out using the memory files which have just been built up (displayed in Appendix I, Part II). Recall that question answering is a by-product of MUSE's candidate selection process, and that action C4 associated with "?" merely activates the response phrase "THE ANSWER TO YOUR QUESTION WAS GIVEN ABOVE.". Sentence 20 illustrates the type of answer given to a "yes-no" question; that is the re-iteration of the input in triple form if it is related to a concept in memory, and the response "I CANNOT INTERPRET THE INPUT." if the input is not so related. Thus, when MUSE reads "car of john be green?", it can relate the input to memory and print out a valid interpretation which means the answer can be taken as "yes". Note that in order to correctly answer the question, MUSE has to substitute VOLKSWAGEN2 as a candidate for "car", and must solve the semantic ambiguity caused by the two meanings for "of".

Sentences 21 and 22 show how candidate units are substituted for "who" and "what" to yield accurate answers to questions. In sentence 21 ("who have green volkswagen?") MUSE first tries WHO1 and PERSON1 as candidate units for "who", but a semantic relationship to memory is found only when JOHN1 is substituted. The next question "ruff chase what car?" has twelve possible

interpretations (since there are four candidates for "what": WHAT1, COLOR1, GREEN1, and RED1; and three candidates for "car": CAR1, VOLKSWAGEN2, and VOLKSWAGEN1) which are generated, tested, and rejected one by one. One of the interesting combinations that is rejected is RUFF1, CHASE1, GREEN1, VOLKSWAGEN2 which becomes the triple set (RUFF1 CHASE1 VOLKSWAGEN2) (VOLKSWAGEN2 COLOR1 GREEN1), the first triple of which is eliminated because it contradicts the memory defined unit-property RUFF1 - (CHASE1 VOLKSWAGEN1). Similarly RUFF1, CHASE1, GREEN1, VOLKSWAGEN1 and RUFF1, CHASE1, RED1, VOLKSWAGEN2 are rejected because of contradictions. Finally the candidate combination RUFF1, CHASE1, RED1, VOLKSWAGEN1 is obtained. This is found to be syntactically correct and breaks into the triples (RUFF1 CHASE1 VOLKSWAGEN1) and (VOLKSWAGEN1 COLOR1 RED1) that intersect with the unit-properties ROVER1 - (CHASE1 VOLKSWAGEN2) and VOLKSWAGEN2 - (COLOR1 GREEN1) respectively. These two intersecting unit-properties were chosen by MUSE because they are semantically related to the triples and are more recently defined than RUFF1 - (CHASE1 VOLKSWAGEN1) and VOLKSWAGEN1 - (COLOR1 RED1). A true question answering procedure would have looked only for the last two unit-properties and would have discarded all others. Thus, MUSE has correctly answered the question, but only

by using inappropriate unit-properties, a fact which truly indicates that in MUSE question answering is a by-product of finding semantic relationships.

Sentences 23, 24, and 25 are examples of the most interesting C-action, the describe action (C6). The command "describe ruff." is interpreted by MUSE and is acted upon to yield the output of sentence 23. Of interest is the lack of commas between the property (CHASE1 VOLKSWAGEN1) and its subproperty (DOWN1 STREET1), and similarly between (CHASE1 VOLKSWAGEN1) and (SPEED1 FAST1). By leaving out the commas, MUSE indicates to the monitor that this is a property and subproperty, not two separate properties. Commas are inserted to distinguish between different properties. Sentence 24, "describe chase.", shows the importance of having defined the third consolidation rule to handle sentences such as "chase be pursue down street.", since the print-out of the memory's concept of "chase" is the same as the English input which defined it originally (see sentence 5 of the sample run). The next input "describe red volkswagen." shows that the describe action does not need to be a two word command, but can be designed to employ all of the model's syntactic and semantic powers to decipher the input. Hence, MUSE

chooses the correct meaning for "volkswagen" from context and performs the describe action on this valid interpretation. Not shown in the sample run are describe actions applied to units with no supersets or subproperties (e.g. OF2 printed out as "OF2 BE A UNIT IN MEMORY."); or with a superset but no subproperties (e.g. THE1 printed out as "THE1 BE A UNIT IN MEMORY: EQUIVALENT1 THE1,.").

The final sentence tells MUSE to "quit processing.", that is to stop without storing any of the memory files. Since the question answering routines and describe actions of this section do not add anything to memory, no loss of information results from quitting rather than halting. The statistics summary shows that question answering has used up a significantly larger amount of time per example (94 seconds/statement) than did the memory build-up sentences of the previous section. This is due mainly to the larger number of candidate units needed to interpret the input. Further considerations of time and space constraints will be included in the next chapter where an evaluation of MUSE is undertaken.

CHAPTER V

EVALUATION OF MUSE

Strengths and Weaknesses of MUSE

The purpose of this chapter is to analyze the structure of MUSE, pointing out the strong parts, describing the weak parts, and proposing some improvements to the model. MUSE has practically no limitations of subject matter; that is there is no reason why the system cannot process information on as many different topics as is desired by the monitor. The model works better, naturally, if its universe of discourse has a small vocabulary, and if as few words as possible have multiple meanings.

Although the input to MUSE is very readable English, there are numerous restrictions to which it must adhere. These restrictions are imposed mainly to enable the easy translation of the input into relational triple notation, and it would not be too difficult to eliminate many of the problems through the use of more sophisticated interpretation procedures. Thus, routines could be designed to separately handle each word of a series of words joined by conjunctions. For example, "ruff and rover be dog." could be

processed as the two sentences "ruff be dog." and "rover be dog.". The handling of modifiers of modifiers (such as "very" in "very little dog") would not be difficult since in the memory notation such modifiers are just subproperties of subproperties. Special syntactic procedures to handle suffixes, tenses, plurals, agreements, etc. could be implemented by associating in some sort of table each such modification of the root word with the word itself. In addition, MUSE could be expanded to include subroutines which would derive information from such items as internal sentence punctuation, capitalized words, and quotation marks around words. Reduction of sentences with intransitive verbs to relational triple notation is more difficult. To see why, note that the sentence "ruff run fast." could be coded as the property (RUFF1 RUN1) with modifying subproperty (SPEED1 FAST1). But this is a property-subproperty unconnected to any memory unit, and hence it means nothing to the system. The question is, therefore, to which unit in memory should the property-subproperty be added? More work needs to be done on this problem before a satisfactory solution can be proposed.

MUSE's memory structure, being similar to that of TLC (Quillian, 1969), is a network designed to store all

manner of data. The structure is able to code extraordinarily complicated English sentences in a simple but elegant manner. The semantic procedures of MUSE use the memory when distinguishing between fine shades of meaning. Much logical ability is also inherent in the memory structure which allows for such items as supersets, subsets, implications (via subsets - i.e. if A is a subset of B, B implies A), and equivalence between words (two words have the same unit).

Among the shortcomings of the memory network is the fact that it is not possible to negate units, to quantify units, or to link units with "or" or "and". However, Quillian (1969) does allow such features in his model (by adding special markers to each unit), so a future implementation of MUSE could be adapted to handle these logical elements. Another possible difficulty with the memory structure is that it is too simple; that is some concepts might have to be distorted in order to adapt them to the memory. Indeed, some words, such as articles, conjunctions, and interjections, have no meaning at all to the system; and ways must be found to code such words in the memory.

MUSE processes its input in a very interesting manner. In some areas of input processing MUSE is highly successful, whereas in other areas it is quite weak. Thus, MUSE makes good use of syntax to learn new

relations, to direct semantic processing, and to help resolve semantic ambiguity; but the syntactic element is itself quite crude, consisting only of a file of parse trees, and even the parse trees are simplified so that the system can comprehend them. What is required is a more sophisticated syntactic component, made up of rewrite rules working on more precise parts of speech. MUSE supplements syntactic processing by using predefined A-actions and B-actions to aid in transforming the input into the internal structure of the model and to transform different paraphrases of the input into the same format. Yet these actions are quite ad hoc, having little or no formal theory behind them. More work needs to be done on the theory behind these actions, and extensive testing of MUSE should be undertaken with a view to discovering the validity of the actions.

MUSE chooses candidate units in a way which yields all possible interpretations of the input; but the model should be able to eliminate "obviously" incorrect candidates before generating all of the candidate combinations. This is necessary to help speed the semantic intersection procedure and to alleviate the very slow processing times that grow even longer as the number of candidate units increases. The model could accomplish

this by "forgetting" unused units or by weighting certain units and links in memory according to their use.

The model does a fairly good job of handling anaphoric references (references to concepts used earlier in text) by coding all input in the same format as the memory structure. Then, when choosing candidates, MUSE picks the most recently defined units first. Under this system, however, a unit will be chosen first less often as other units are defined, even if the unit is used again in a more recent context. What is needed to remedy this is a procedure to cycle units to the beginning of the unit file whenever they occur in the input, thus insuring that the criterion of choice is recency of use, not recency of definition.

MUSE uses semantic intersection techniques to discern the context of words in the input and to eliminate syntactic ambiguities. These techniques have been tested in both TLC and MUSE and work for a wide variety of sentences, but there still remain problems. Sometimes intersections are too powerful and lead to wrong interpretations; and sometimes they are not strong enough, and valid comparisons are not made. The artificial limiting of superset chains to three links should be removed when conditions are somehow "right" for proceeding further. Finally, the mainly breadth-first

search of TLC has been destroyed in MUSE by directing semantic processing using syntax (that is, the model examines all possible intersections from one candidate combination before proceeding to the next combination). Since syntax directed semantic processing is useful in cutting down processing time, some compromise will probably have to be reached to reinstate more parallel searching. Perhaps this could take the form of intersecting each candidate combination to only one level of superset before proceeding to the next combination. If no valid interpretations are obtained after all combinations have been tried, the model proceeds to the next level of intersection for each candidate combination, and so on until a valid interpretation is discovered.

MUSE answers questions posed in English, but only as a by-product of the semantic intersection techniques. To expand MUSE into a truly deductive question answering system, special routines would have to be designed to enable the model to handle questions in a different manner than other sentences. Thus, it would not be difficult to get the model to search for candidates of "who", "what", "where", and "when" with the specific purpose of finding a fact in memory exactly matching the question. Also, procedures could easily be defined

to yield "yes" or "no" answers to true or false questions, rather than just repeating the question if it is true.

The motor actions (C-actions) in MUSE enable the model to obey orders and perform tasks, something which very few non-procedure oriented natural language models can do (however, the procedure oriented systems of such researchers as Woods, 1968; Kellogg, 1968; and Kochen, 1969 are based on performing executive routines). New, more complex actions could be devised to do such things as forgetting concepts, finding synonyms of memory words, or performing intricate deductions from input. Potential improvements in existing actions range from modifying the output format of the describe action to a complete revamping of the update action to allow memory items to be added more quickly.

MUSE's output consists of simple English triples which are made a little less readable by the appending of numbers to the end (indicating which meaning is meant in the current context) and by the printing of the intersecting unit-properties from memory. These aspects could be eliminated to improve readability. The addition of a generative grammar to the model, although not an easy task, could improve the sophistication of the output.

MUSE is very easy to expand. The sample run of the last chapter shows how the model learned fourteen new words, fifteen new units, eleven new properties, and one new parse tree. An improvement on TLC is the way that new units are defined only when necessary and that all discovered properties and supersets are added to old units in memory. In addition, MUSE can learn facts not already in memory by creating new properties, rather than just by adapting old properties as is done in TLC. MUSE can also learn from example and recognize paraphrases of old facts in memory. Finally, duplication of words, units, and properties is kept to a minimum wherever possible.

Among the undesirable features of this learning procedure is that new words must be flagged with a marker containing a part of speech symbol and any associated actions. To eliminate such flagging, the model could define a completely new word only when it comes across a word not in the dictionary and could define a new unit for an old word only when syntactic or semantic procedures break down on all the old units for that word. Complex procedures could be developed to derive from usage the part of speech of a unit. Under this system actions would be associated with the parts of speech, not the units, so the assigning of a

part of speech would also assign an action (naturally, this excludes C-actions). Thus, a much finer distinction between parts of speech would be necessary.

The way the model discovers relations not yet coded in memory is by unstacking nearly correct interpretations from the failed triple stack; not a very satisfactory method. Some sort of recognition procedure for new concepts must be devised. There is no way of adding new actions to MUSE without programming new subroutines, although new pointers to existing actions can easily be added. It seems unlikely that self modifying programs will be possible in the near future, so the automatic addition of actions is something that is a long way off.

MUSE is presently programmed in the string manipulation language SNOBOL4. SNOBOL4 allows the discovery of patterns in the input, making it easy to recognize input words and elements in the memory files. The major problem with the current implementation of SNOBOL4 is that it runs interpretively, that is it translates each statement line by line. This factor, when combined with the interactive nature of MTS, means that processing time is very slow. Although the procedures of MUSE have been designed to save as much time as possible, there are many inefficiencies in the program itself (due to the conversion of the program to MTS from another time

sharing system, CP/CMS) slowing still further the speed of the model. These inefficiencies could be overcome by an improved version of SNOBOL4 and a thorough revision of the program.

Conclusion

MUSE is a natural language model that is based strongly on Quillian's (1969) TLC. Although its simple English input is somewhat more restricted than that of TLC, MUSE surpasses Quillian's model in many areas, especially syntactic processing. MUSE uses a file of parse trees to help eliminate semantic ambiguity, to direct semantic processing, and to aid in the derivation of new properties. MUSE learns new words and units from on-line discourse through a none too satisfactory method of flagging new concepts in the input sentence. New properties are derived most often by using syntactic procedures and a failed triple stack. The memory size is kept small by never adding redundant words, units, or properties to memory. Syntax oriented executive actions which help the model to interpret the input and aid in the uniform coding of paraphrases can be appended to units in memory. In addition, motor actions, associated with some memory units, are defined to allow MUSE to act on the various interpretations given to the

input. The output from MUSE is simple English triples or predefined English sentences. Implemented on a time sharing system, the model interacts with the monitor, thus allowing him to keep track of what is going on. Many improvements to MUSE have been suggested in this chapter; and further testing will likely indicate that still more changes would benefit the model. Even as it is presently implemented, however, MUSE accomplishes enough to warrant considerable optimism for the future of computer processing of natural language.

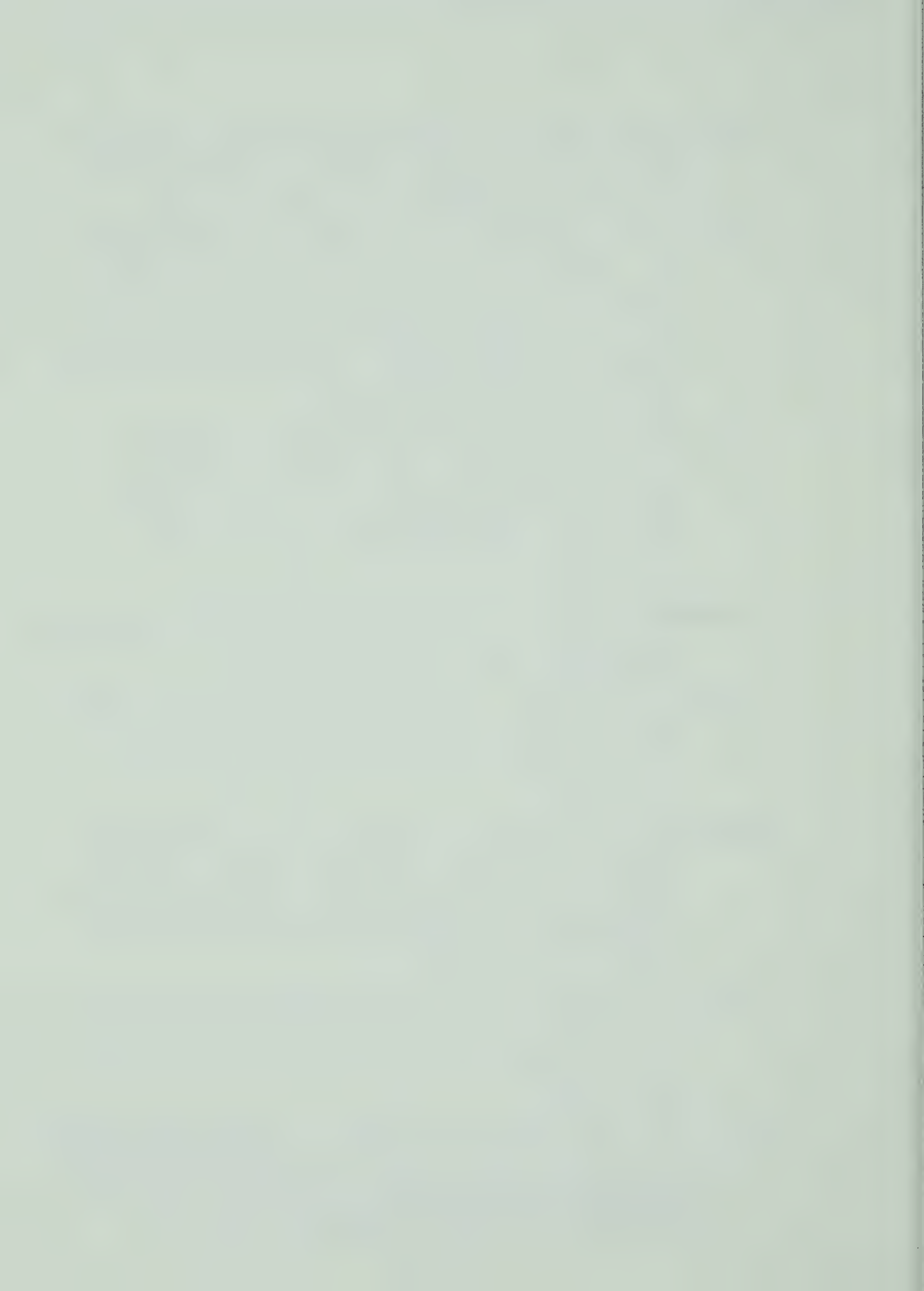
REFERENCES

- Abelson, R.P. and Reich, C.M. (1969). "Implicational Molecules: A Method for Extracting Meaning from Input Sentences". I.J.C.A.I.^{*}, Washington, D.C., pp. 641-648.
- Becker, J.D. (1969). "The Modelling of Simple Analogic and Inductive Processes in a Semantic Memory System". I.J.C.A.I., Washington, D.C., pp.655-668.
- Bobrow, D.G. (1968). "Natural Language Input for a Computer Problem-Solving System". In Semantic Information Processing, Minsky (ed.), pp.135-215.
- Charniak, E. (1969). "Computer Solution of Calculus Word Problems". I.J.C.A.I., Washington, D.C., pp. 303-316.
- Chomsky, N. (1965). Aspects of the Theory of Syntax. The M.I.T. Press, M.I.T., Cambridge, Mass.
- Colby, K.M. and Smith, D.C. (1969). "Dialogues Between Humans and an Artificial Belief System". I.J.C.A.I., Washington, D.C., pp. 319-324.
- Colby, K.M., Tesler, L. and Enea, H. (1969). "Experiments with a Search Algorithm for the Data Base of a Human Belief System". I.J.C.A.I., Washington, D.C., pp. 649-654.

* Proceedings of the International Joint Conference on Artificial Intelligence.



- Coles, L.S. (1969). "Talking with a Robot in English".
I.J.C.A.I., Washington, D.C., pp. 587-596,
- Craig, J.A., Berezner, S.C., Carney, H.C. and Longyear, C.R. (1966). "DEACON: Direct English Access and Control". AFIPS Conference Proceedings - 1966 Fall Joint Comput. Conf., Vol. 29, Spartan Books, Washington, D.C., pp. 365-380.
- Dewar, H., Bratley, P. and Thorne, J.P. (1969). "A Program for the Syntactic Analysis of English Sentences". Comm. ACM 12, 8 (August 1969), pp. 476-479.
- Feigenbaum, E.A. and Feldman, J. (eds.) (1963). Computers and Thought. McGraw-Hill, New York.
- Friedman, J. (1969). "A Computer System for Transformational Grammar". Comm. ACM 12, 6 (June 1969), pp. 341-348.
- Green, B.F., Wolf, A.K., Chomsky, C. and Laughery, K. (1963). "BASEBALL: An Automatic Question Answer". In Computers and Thought, Feigenbaum and Feldman (eds.), pp. 207-216.
- Green, C. (1969). "Application of Theorem Proving to Problem Solving". I.J.C.A.I., Washington, D.C., pp. 219-240.
- Katz, J.J. and Postal, P.M. (1964). An Integrated Theory of Linguistic Descriptions. Research Monograph #26, The M.I.T. Press, Cambridge, Mass.



- Kellogg, C.H. (1968). "A Natural Language Compiler for On-Line Data Management". AFIPS Conference Proceedings - 1968 Fall Joint Comput. Conf., Vol. 33, Thompson Book Co., Washington, D.C., pp. 473-492.
- Kochen, M. (1969). "Automatic Question Answering of English-Like Questions About Simple Diagrams". J. ACM 16, 1 (January 1969), pp. 26-48.
- Langacker, D.W. (1967). Language and Its Structure: Some Fundamental Linguistic Concepts. Harcourt, Brace and World Inc., New York.
- Lindsay, R.K. (1963). "Inferential Memory as the Basis of Machines which Understand Natural Language". In Computers and Thought, Feigenbaum and Feldman (eds.), pp. 217-233.
- Minsky, M. (ed.) (1968). Semantic Information Processing. The M.I.T. Press, M.I.T., Cambridge, Mass.
- Quillian, M.R. (1966). Semantic Memory. Unpublished doctoral dissertation, Carnegie Institute of Technology.
- Quillian, M.R. (1967). "Word Concepts: A Theory and Simulation of Some Basic Semantic Capabilities". Behav. Sci. 12, pp. 410-429.
- Quillian, M.R. (1968). "Semantic Memory". In Semantic Information Processing, Minsky (ed.), pp. 216-270.



- Quillian, M.R. (1969). "The Teachable Language Comprehender: A Simulation Program and Theory of Language". Comm. ACM 12, 8 (August 1969), pp. 459-476.
- Raphael, B. (1968). "SIR: Semantic Information Retrieval". In Semantic Information Processing, Minsky (ed.), pp. 33-134.
- Rosenbaum, P.S. (1967). "A Grammar Base Question-Answering Procedure". Comm. ACM 10, 10 (October 1967), pp. 630-635.
- Schank, R.C., Tesler, L. and Weber, S. (1970). SPINOZA II: Conceptual Case-Based Natural Language Analysis. Memo AIM-109, Stanford Artificial Intelligence Project, Stanford, California.
- Schwarz, R.M., Burger, J.F. and Simmons, R.F. (1970). "A Deductive Question-Answer for Natural Language Inference". Comm. ACM 13, 3 (March 1970), pp. 167-183.
- Shapiro, S.C. and Woodmansee, G.H. (1969). "A Net Structure Based Relational Question Answerer: Description and Examples". I.J.C.A.I., Washington, D.C., pp. 325-346.
- Simmons, R.F. (1970). "Natural Language Question Answering Systems: 1969". Comm. ACM 13, 1 (January 1970), pp. 15-30.
- Tharp, A.L. and Krulee, G.K. (1969). "Using Relational Operators to Structure Long-Term Memory". I.J.C.A.I., Washington, D.C., pp. 579-586.

Weizenbaum, J. (1966). "ELIZA - A Computer Program for the Study of Natural Language Communications Between Man and Machine". Comm. ACM 9, 1 (January 1966), pp. 36-45.

Woods, W.A. (1968). "Procedural Semantics for a Question-Answering Machine". AFIPS Conference Proceedings - 1968 Fall Joint Comput. Conf., Vol. 33, Thompson Book Co., Washington, D.C., pp. 457-471.

MEMORY FILES

Part I: Files Before Memory Build Up

```

$!list dictionp
> 1 31 . GREEN;U33,. RED;U32,. COLOR;U31,. HAVE;U29,. OF;U30,U28,. ALSO;U27,. THAT;
> 2 U26,. ANOTHER;U25,. THIS;U24,. SPEED;U23,. FAST;U22,U21,. PROCESSING;U20,. DIC
> 3 TION;U19,. PFILE;U18,. UFILE;U17,. SYNTAX;U16,. PERSON;U15,. THING;U14,. CHANG
> 4 E;U13,. QUIT;U12,. DESCRIBE;U11,. DEFINE;U10,. EQUIVALENT;U9,. UPDATE;U8,. HAL
> 5 T;U7,. BE;U6,. WHO;U5,. WHAT;U4,. MUSE;U3,. YOU;U2,. SELF;U1,.:
#$!list ufilep
> 1 33 . U33;D;A3,. U31;P0,. U32;D;A3,. U31;P0,. U31;N;. U4;P0,. U30;T;B2,. U0;P0,. U
> 2 29;V;. U0;P0,. U28;T;B2,. U29;P0,. U27;B;A1,. U0;P0,. U26;C;A1,. U0;P0,. U25;R;A
> 3 1,. U0;P0,. U24;R;A1,. U0;P0,. U23;N;. U0;P0,. U22;B;A3,. U23;P0,. U21;N;. U0;P0,
> 4 . U20;N;. U14;P0,. U19;N;. U14;P0,. U18;N;. U14;P0,. U17;N;. U14;P0,. U16;N;. U14;
> 5 P0,. U15;N;. U5;P0,. U14;N;. U0;P0,. U13;V;A2,C8,. U0;P0,. U12;V;A2,C7,. U0;P0,.
> 6 U11;V;A2,C6,. U0;P0,. U10;V;A2,C5,. U0;P0,. U9;V;C3,. U0;P0,. U8;V;A2,C2,. U0;P
> 7 0,. U7;V;A2,C1,. U0;P0,. U6;E;B1,. U0;P0,. U5;C;. U0;P0,. U4;C;. U0;P0,. U3;N;. U1
> 8 ;P0,. U2;P;. U1;P0,. U1;N;. U0;P6,P5,P4,P3,P2,P1,.:
#$!list pfilep
> 1 6 . P5;U11;U1;P0,. P5;U13;U14;P0,. P4;U12;U14;P0,. P3;U10;U14;P0,. P2;U8;U14;
> 2 P0,. P1;U7;U14;P0,.:
#$!list syntaxp
> 1 14 . (6)NPVEBDRTICJ,VE,NPVEBDRTICJ,C,VE,NPVEBDRTICJ,+S1;Z1,V1-V1;Z2,N1-N1
> 2 ;Z3,S2-S2;Z4,V2-V2;Z5,Z6-. (8)NP,VE,R,D,NP,T,R,NP,+S1;Z1,V1-V1;Z2,N1-N1;Z
> 3 3,N2-N2;N3,T1-N3;Z5,D1-D1;Z0,Z4-T1;Z6,N4-N4;Z7,Z8-. (8)NP,VE,R,D,NP,T,R,N
> 4 P,+S1;Z1,V1-V1;V2,H2-V2;Z2,T1-T1;Z6,N1-N1;Z7,Z8-N2;N3,D1-N3;Z3,Z5-D1;Z0,
> 5 Z4-. (5)NP,VE,R,D,NP,+S1;Z1,V1-V1;Z2,N1-N1;Z3,Z5-D1;Z0,Z4-. (6)R,
> 6 NP,VE,R,NP,B,+S1;N1,V1-N1;Z1,Z2-V1;V2,N2-N2;Z4,Z5-V2;Z3,Z6-. (4)NP,VE,R,N
> 7 P,+S1;Z1,V1-V1;Z2,N1-N1;Z3,Z4-. (4)NP,VE,R,D,NP,+S1;Z1,V1-V1;Z2,N1-N1;Z4,D1
> 8 -D1;Z0,Z3-. (3)VE,D,NP,+S1;Z0,V1-V1;Z1,N1-N1;Z3,D1-D1;Z0,Z2-. (2)VE,NPVEBD
> 9 RTICJ,+S1;Z0,V1-V1;Z1,Z2-. (3)NPVEBDRTICJ,VE,NPVEBDRTICJ,+S1;Z1,V1-V1;Z2,
> 10 Z3-. (3)NP,VE,D,+S1;Z1,V1-V1;Z2,D1-D1;Z0,Z3-. (5)NP,T,NP,VE,D,+S1;N1,V1-N1
> 11 ;Z1,T1-T1;Z2,Z3-V1;Z4,D1-D1;Z0,Z5-. (6)NP,T,R,NP,VE,D,+S1;N1,V1-N1;Z1,T1-
> 12 T1;Z2,N2-N2;Z3,Z4-V1;Z5,D1-D1;Z0,Z6-. (5)NP,V,R,NP,B,+S1;Z1,V1-V1;V2,N1-V
> 13 2;Z2,B1-B1;Z0,Z5-N1;Z3,Z4-.:

```


Part II: Files After Memory Build Up

```

$!list diction
> 45 .ROVER;U48,.JOHN;U46,.CAR;U45,.VOLKSWAGEN;U47,U44,.ANIMAL;U43,.RUFF;U
> 42,.DCG;U41,.STREET;U40,.DOWN;U39,.PURSUE;U38,.CHASE;U37,.THE;U36,.A;U36
> .MEANS;U34,.GREEN;U33,.RED;U32,.COLOR;U31,.HAVE;U29,.OF;U30,U28,.ALSO;U
> 27,.THAT;U26,.ANOTHER;U25,.THIS;U24,.SPEED;U23,.FAST;U22,U21,.PROCESSING
> ;U20,.DICTION;U19,.PFILE;U18,.UFILE;U17,.SYNTAX;U16,.PERSON;U15,.THING;U
> 14,.CHANGE;U13,.QUIT;U12,.DESCRIBE;U11,.DEFINE;U10,.EQUIVALENT;U9,.UPDAT
> E;U8,.HALT;U7,.BE;U6,.WHO;U5,.WHAT;U4,.MUSE;U3,.YOU;U2,.SELF;U1,.:
#$!list ufile
> 48 .U48;N;:U41;P17,.U47;N;:U45;P16,.U46;N;:U15;P15,.U45;N;:U0;P0,.U44;N;
> ;U45;P11,.U43;N;:U0;P0,.U42;N;:U41;P14,P12,.U41;N;:U43;P0,.U40;N;:U0;P0,
> .U39;T;:U0;P0,.U38;V;:U0;P0,.U37;V;:U38;P9,.U36;R;A1,.U0;P8,.U34;E;B1,:U
> 0;P0,.U33;D;A3,.U31;P0,.U32;D;A3,.U31;P0,.U31;N;:U4;P0,.U30;T;B2,:U0;P0,
> .U29;V;:U0;P0,.U28;T;B2,:U29;P0,.U27;E;A1,.U0;P0,.U26;C;A1,.U0;P0,.U25;R
> ;A1,:U0;P0,.U24;R;A1,:U0;P0,.U23;N;:U0;P0,.U22;E;A3,:U23;P0,.U21;N;:U0;P
> 0,.U20;N;:U14;P0,.U19;N;:U14;P0,.U18;N;:U14;P0,.U17;N;:U14;P0,.U16;N;:U1
> 4;P0,.U15;N;:U5;P0,.U14;N;:U0;P0,.U13;V;A2,C8,:U0;P0,.U12;V;A2,C7,:U0;P0
> ,.U11;V;A2,C6,:U0;P0,.U10;V;A2,C5,:U0;P0,.U9;V;C3,:U0;P0,.U8;V;A2,C2,:U0
> ;P0,.U7;V;A2,C1,:U0;P0,.U6;E;D1,:U0;P0,.U5;C;:U0;P0,.U4;C;:U0;P0,.U3;N;:
> U1;P0,.U2;P;:U1;P0,.U1;N;:U0;P10,P7,P6,P5,P4,P3,P2,P1,.:
#$!list pfile
> 17 .P17;U37;U47;P0,.P16;U31;U33;P0,.P15;U29;U47;P0,.P14;U37;U44;P13,.P13
> ;U23;U22;P0,.P12;U37;U44;P9,.P11;U31;U32;P0,.P10;U10;U41;P0,.P9;U39;U40;
> P0,.P8;U9;U36;P0,.P7;U11;U14;P0,.P6;U11;U1;P0,.P5;U13;U14;P0,.P4;U12;U14
> ;P0,.P3;U10;U14;P0,.P2;U8;U14;P0,.P1;U7;U14;P0,.:
#$!list syntax
> 15 .(5)NPVEBDRTICJ,VE,NPVEBDRTICJ,T,NP,+S1;Z1,V1-V1;Z2,N1-N1;Z3,T1-T1;Z4
> ,Z5-.(6)NPVEBDRTICJ,VE,NPVEBDRTICJ,C,VE,NPVEBDRTICJ,+S1;Z1,V1-V1;Z2,N1-N
> 1;Z3,S2-S2;Z4,V2-V2;Z5,Z6-.(8)NP,VE,R,D,NP,T,R,NP,+S1;Z1,V1-V1;Z2,N1-N1;
> Z3,N2-N2;N3,T1-N3;Z5,D1-D1;Z0,Z4-T1;Z6,N4-N4;Z7,Z8-.(8)NP,VE,R,D,NP,T,R,
> NP,+S1;Z1,V1-V1;V2,N2-V2;Z2,T1-T1;Z6,N1-N1;Z7,Z8-N2;N3,D1-N3;Z3,Z5-D1;Z0
> ,Z4-.(5)NP,VE,R,D,NP,+S1;Z1,V1-V1;Z2,N1-N1;N2,D1-N2;Z3,Z5-D1;Z0,Z4-.(6)R
> ,NP,VE,R,NP,B,+S1;N1,V1-N1;Z1,Z2-V1;V2,N2-N2;Z4,Z5-V2;Z3,Z6-.(4)NP,VE,R,
> NP,+S1;Z1,V1-V1;Z2,N1-N1;Z3,Z4-.(4)NP,VE,R,D,NP,+S1;Z1,V1-V1;Z2,N1-N1;Z4,D
> 1-D1;Z0,Z3-.(3)VE,D,NP,+S1;Z0,V1-V1;Z1,N1-N1;Z3,D1-D1;Z0,Z2-.(2)VE,NPVEB
> DRTICJ,+S1;Z0,V1-V1;Z1,Z2-.(3)NPVEBDRTICJ,VE,NPVEBDRTICJ,+S1;Z1,V1-V1;Z2
> ,Z3-.(3)NP,VE,D,+S1;Z1,V1-V1;Z2,D1-D1;Z0,Z3-.(5)NP,T,NP,VE,D,+S1;N1,V1-N
> 1;Z1,T1-T1;Z2,Z3-V1;Z4,D1-D1;Z0,Z5-.(6)NP,T,R,NP,VE,D,+S1;N1,V1-N1;Z1,T1
> -T1;Z2,N2-N2;Z3,Z4-V1;Z5,D1-D1;Z0,Z6-.(5)NP,V,R,NP,B,+S1;Z1,V1-V1;Z2,N1-
> V2;Z2,B1-B1;Z0,Z5-N1;Z3,Z4-.:

```


APPENDIX II

SAMPLE RUN

Part I: Memory Build Up

```
#SRUN *SNOBOL4 1=DICTION 2=UFILE 3=PFILE 4=SYNTAX 5=WURK3+
  *NSOURCE* 6=-LISTING(1,1172)+*SINK* 8=TRCFILE PAR=SIZE=75
#EXECUTION BEGINS
```

0 SYNTACTIC ERROR(S) IN SOURCE PROGRAM

GOOD MORNING. MY NAME IS MUSE.

PLEASE TYPE YOUR INPUT.

1. update syntax.

THE INPUT CAN BE INTERPRETED AS FOLLOWS:

SELF1 UPDATE1 SYNTAX1 (SELF1 UPDATE1 THING1).

IS THIS A VALID INTERPRETATION?

yes

DO YOU WANT THIS ADDED TO MEMORY?

no

DO YOU WISH ME TO ACT, IF POSSIBLE?

yes

SYNTAX UPDATE ENTERED.

WRITE LENGTH OF PART OF SPEECH VECTOR.

5

WRITE PART OF SPEECH VECTOR.

npvebdrticj,ve,npvebdrticj,t,np,

WRITE PARSE.

s1;z1,v1-v1;z2,n1-n1;z3,t1-t1;z4,z5-

SYNTAX UPDATE ENTERED.

WRITE LENGTH OF PART OF SPEECH VECTOR.

:

DO YOU WANT TO CONTINUE UPDATE?

yes

DICTION UPDATE ENTERED.

WRITE WORD.

means

DICTION UPDATE ENTERED.

WRITE WORD.

:

UFILE UPDATE ENTERED.

WRITE PART OF SPEECH.

e

WRITE ACTIONS.

b1,

WRITE SUPERSSET.

u0

WRITE SUBPROPERTIES.

p0,

UFILE UPDATE ENTERED.

WRITE PART OF SPEECH.

:


```

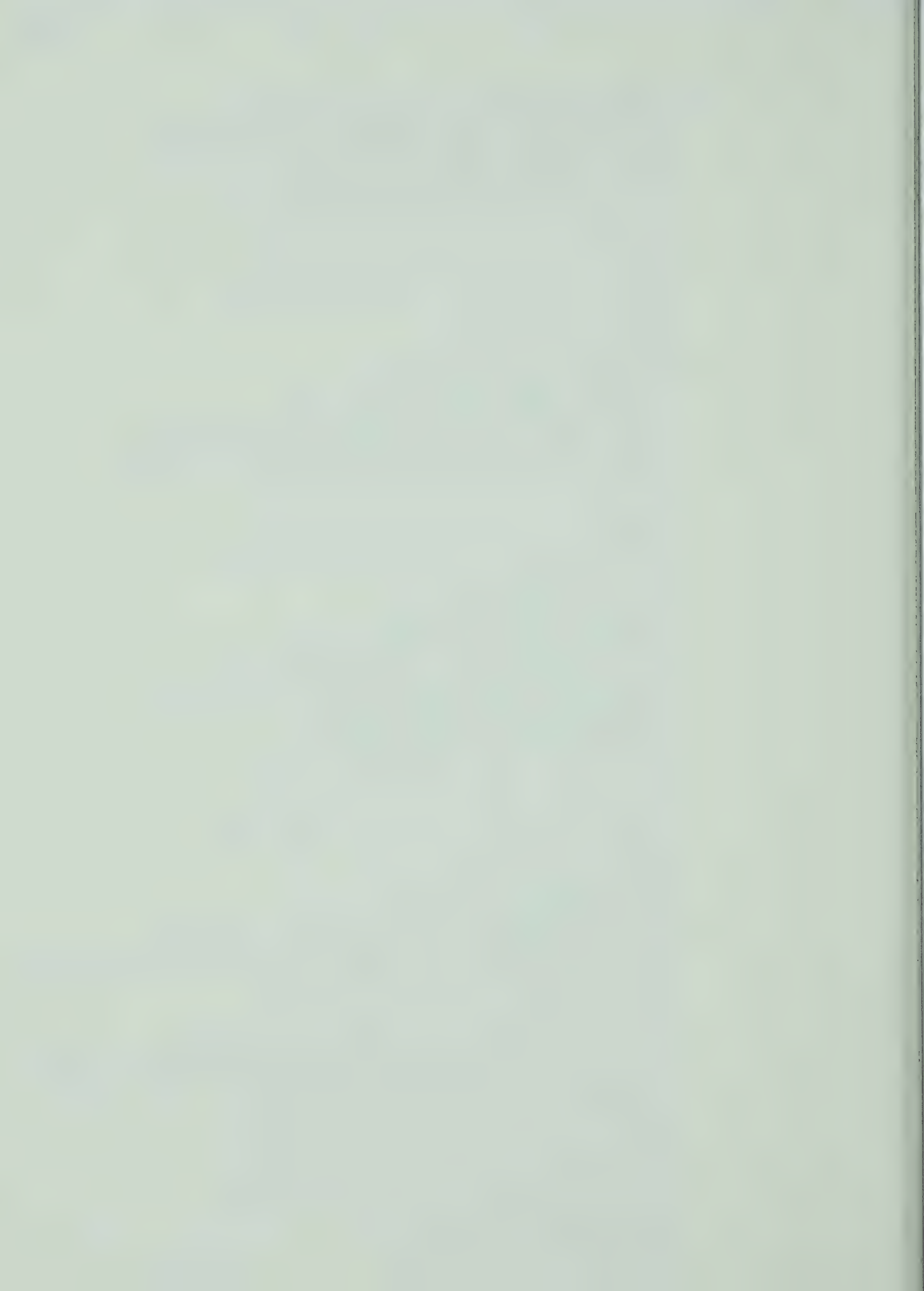
PFILE UPDATE ENTERED.
WRITE ATTRIBUTE.
u10
WRITE VALUE.
u14
WRITE SUBPROPERTIES.
p0,
PFILE UPDATE ENTERED.
WRITE ATTRIBUTE.
:
SYNTAX UPDATE ENTERED.
WRITE LENGTH OF PART OF SPEECH VECTOR.
:
DO YOU WANT TO CONTINUE UPDATE?
no
PLEASE TYPE YOUR INPUT.
2. muse change diction.
THE INPUT CAN BE INTERPRETED AS FOLLOWS:
MUSE1 CHANGE1 DICTION1 (SELF1 CHANGE1 THING1 ).
IS THIS A VALID INTERPRETATION?
yes
DO YOU WANT THIS ADDED TO MEMORY?
no
DO YOU WISH ME TO ACT, IF POSSIBLE?
yes
DICTION CHANGE ENTERED.
ENTER ELEMENT TO BE CHANGED.
means
MEENS WAS .MEENS;U34,
ENTER NEW VERSION.
.means;u34,
ENTER ELEMENT TO BE CHANGED.
:
UFILE CHANGE ENTERED.
ENTER ELEMENT TO BE CHANGED.
u1
U1 WAS .U1;N;;;U0;P6,P5,P4,P3,P2,P1,
ENTER NEW VERSION.
.u1;n;;;u0;p7,p6,p5,p4,p3,p2,p1,
ENTER ELEMENT TO BE CHANGED.
:
PFILE CHANGE ENTERED.
ENTER ELEMENT TO BE CHANGED.
p7
P7 WAS .P7;U10;U14;P0,
ENTER NEW VERSION.
.p7;u11;u14;p0,
ENTER ELEMENT TO BE CHANGED.
:
DO YOU WANT TO CONTINUE CHANGE?
no

```


3. PLEASE TYPE YOUR INPUT.
 person change pfile.
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 PERSON1 CHANGE1 PFILE1 ().
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 no
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 NO ACTION POSSIBLE.
 PLEASE TYPE YOUR INPUT.
4. a(r;a1,) equivalent the(r;a1,).
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 A1 EQUIVALENT1 THE1 ().
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 EQUIVALENCE DEFINED.
 PLEASE TYPE YOUR INPUT.
5. chase(v;) means pursue(v;) down(t;) street(n;).
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 CHASE1 BE PURSUE1 (PURSUE1 SUPERSET-OF CHASE1).
 CHASE1 DOWN1 STREET1 ().
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 NO ACTION POSSIBLE.
 PLEASE TYPE YOUR INPUT.
6. define dog(n;).
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 SELF1 DEFINE1 DOG1 ().
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 THE CONCEPT HAS BEEN DEFINED.
 PLEASE TYPE YOUR INPUT.

7. ruff(n;) be dog that be animal(n;).
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 RUFF1 BE DOG1 (DOG1 SUPERSET-OF RUFF1).
 DOG1 BE ANIMAL1 (ANIMAL1 SUPERSET-OF DOG1).
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 NO ACTION POSSIBLE.
 PLEASE TYPE YOUR INPUT.
8. volkswagen(n;) be car(n;).
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 VOLKSWAGEN1 BE CAR1 (CAR1 SUPERSET-OF VOLKSWAGEN1).
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 NO ACTION POSSIBLE.
 PLEASE TYPE YOUR INPUT.
9. volkswagen be red.
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 VOLKSWAGEN1 COLOR1 RED1 ().
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 NO ACTION POSSIBLE.
 PLEASE TYPE YOUR INPUT.
10. ruff chase the red volkswagen down the street.
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 RUFF1 CHASE1 VOLKSWAGEN1 ().
 CHASE1 DOWN1 STREET1 ().
 VOLKSWAGEN1 COLOR1 RED1 ().
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 NO ACTION POSSIBLE.
 PLEASE TYPE YOUR INPUT.

11. ruff chase the volkswagen fast.
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 RUFF1 CHASE1 VOLKSWAGEN1 ().
 CHASE1 SPEED1 FAST2 ().
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 NO ACTION POSSIBLE.
 PLEASE TYPE YOUR INPUT.
12. john(n;) be person.
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 JOHN1 BE PERSON1 (PERSON1 SUPERSET-OF JOHN1).
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 NO ACTION POSSIBLE.
 PLEASE TYPE YOUR INPUT.
13. john have a green volkswagen(n;).
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 JOHN1 HAVE1 VOLKSWAGEN2 ().
 VOLKSWAGEN2 COLOR1 GREEN1 ().
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 NO ACTION POSSIBLE.
 PLEASE TYPE YOUR INPUT.
14. this volkswagen be a car also.
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 VOLKSWAGEN1 BE CAR1 (CAR1 SUPERSET-OF VOLKSWAGEN1).
 IS THIS A VALID INTERPRETATION?
 no
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 VOLKSWAGEN2 BE CAR1 (CAR1 SUPERSET-OF VOLKSWAGEN2).
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 NO ACTION POSSIBLE.
 PLEASE TYPE YOUR INPUT.



15. color of the volkswagen be green.
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 VOLKSWAGEN2 COLOR1 GREEN1 (VOLKSWAGEN2 COLOR1 GREEN1).
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 no
 PLEASE TYPE YOUR INPUT.
16. rover(n;) be another dog.
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 ROVER1 BE DOG1 (DOG1 SUPERSET-OF ROVER1).
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 NO ACTION POSSIBLE.
 PLEASE TYPE YOUR INPUT.
17. rover chase the green volkswagen.
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 ROVER1 CHASE1 VOLKSWAGEN2 (RUFF1 CHASE1 VOLKSWAGEN1).
 VOLKSWAGEN2 COLOR1 GREEN1 (VOLKSWAGEN2 COLOR1 GREEN1).
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 yes
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 NO ACTION POSSIBLE.
 PLEASE TYPE YOUR INPUT.
18. john chase dog be.
 I CANNOT INTERPRET THE INPUT.
 I AM PROCEEDING TO THE NEXT STATEMENT.
 PLEASE TYPE YOUR INPUT.
19. halt processing.
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 SELF1 HALT1 PROCESSING1 (SELF1 HALT1 THING1).
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 no
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 THANKS FOR THE CONVERSATION.

NORMAL TERMINATION AT LEVEL 0
LAST STATEMENT EXECUTED WAS 339

SNOBOL4 STATISTICS SUMMARY-

24961 MS. COMPILATION TIME

947643 MS. EXECUTION TIME

62617 STATEMENTS EXECUTED, 13064 FAILED

4199 ARITHMETIC OPERATIONS PERFORMED

46639 PATTERN MATCHES PERFORMED

306 REGENERATIONS OF DYNAMIC STORAGE

15.134 MS. AVERAGE PER STATEMENT

#EXECUTION TERMINATED
#

Part II: Question Answering

```
#$RUN *SNOBOL4 1=DICTION 2=UFILE 3=PFILE 4=SYNTAX 5=WURK3+
  *MSOURCE* 6=-LISTING(1,1172)+*SINK* 8=TRCFILE PAR=SIZE=75
#EXECUTION BEGINS
```

```
0 SYNTACTIC ERROR(S) IN SOURCE PROGRAM
```

- GOOD MORNING. MY NAME IS MUSE.
PLEASE TYPE YOUR INPUT.
20. car of john be green?
THE INPUT CAN BE INTERPRETED AS FOLLOWS:
JOHN1 HAVE1 VOLKSWAGEN2 (JOHN1 HAVE1 VOLKSWAGEN2).
VOLKSWAGEN2 COLOR1 GREEN1 (VOLKSWAGEN2 COLOR1 GREEN1).
IS THIS A VALID INTERPRETATION?
yes
DO YOU WANT THIS ADDED TO MEMORY?
no
DO YOU WISH ME TO ACT, IF POSSIBLE?
yes
THE ANSWER TO YOUR QUESTION WAS GIVEN ABOVE.
PLEASE TYPE YOUR INPUT.
21. who have green volkswagen?
THE INPUT CAN BE INTERPRETED AS FOLLOWS:
JOHN1 HAVE1 VOLKSWAGEN2 (JOHN1 HAVE1 VOLKSWAGEN2).
VOLKSWAGEN2 COLOR1 GREEN1 (VOLKSWAGEN2 COLOR1 GREEN1).
IS THIS A VALID INTERPRETATION?
yes
DO YOU WANT THIS ADDED TO MEMORY?
no
DO YOU WISH ME TO ACT, IF POSSIBLE?
yes
THE ANSWER TO YOUR QUESTION WAS GIVEN ABOVE.
PLEASE TYPE YOUR INPUT.
22. ruff chase what car?
THE INPUT CAN BE INTERPRETED AS FOLLOWS:
RUFF1 CHASE1 VOLKSWAGEN1 (ROVER1 CHASE1 VOLKSWAGEN2).
VOLKSWAGEN1 COLOR1 RED1 (VOLKSWAGEN2 COLOR1 GREEN1).
IS THIS A VALID INTERPRETATION?
yes
DO YOU WANT THIS ADDED TO MEMORY?
no
DO YOU WISH ME TO ACT, IF POSSIBLE?
yes
THE ANSWER TO YOUR QUESTION WAS GIVEN ABOVE.
PLEASE TYPE YOUR INPUT.

23. describe ruff.
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 SELF1 DESCRIBE1 RUFF1 ().
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 no
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 RUFF1 BE A UNIT IN MEMORY.
 RUFF1 BE DOG1 : CHASE1 VOLKSWAGEN1 SPEED1 FAST2
 , CHASE1 VOLKSWAGEN1 DOWN1 STREET1 , .
 PLEASE TYPE YOUR INPUT.
24. describe chase.
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 SELF1 DESCRIBE1 CHASE1 ().
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 no
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 CHASE1 BE A UNIT IN MEMORY.
 CHASE1 BE PURSUE1 : DOWN1 STREET1 , .
 PLEASE TYPE YOUR INPUT.
25. describe red volkswagen.
 I AM NOT SURE BUT I THINK THAT
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 SELF1 DESCRIBE1 VOLKSWAGEN1 ().
 VOLKSWAGEN1 COLOR1 RED1 ().
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 no
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 VOLKSWAGEN1 BE A UNIT IN MEMORY.
 VOLKSWAGEN1 BE CAR1 : COLOR1 RED1 , .
 PLEASE TYPE YOUR INPUT.
26. quit processing.
 THE INPUT CAN BE INTERPRETED AS FOLLOWS:
 SELF1 QUIT1 PROCESSING1 (SELF1 QUIT1 THING1).
 IS THIS A VALID INTERPRETATION?
 yes
 DO YOU WANT THIS ADDED TO MEMORY?
 no
 DO YOU WISH ME TO ACT, IF POSSIBLE?
 yes
 THANKS FOR THE CONVERSATION.

NORMAL TERMINATION AT LEVEL 0
LAST STATEMENT EXECUTED WAS 395

SNOBOL4 STATISTICS SUMMARY-

24869 MS. COMPILATION TIME

656788 MS. EXECUTION TIME

41213 STATEMENTS EXECUTED, 9163 FAILED

2354 ARITHMETIC OPERATIONS PERFORMED

32603 PATTERN MATCHES PERFORMED

231 REGENERATIONS OF DYNAMIC STORAGE

15.936 MS. AVERAGE PER STATEMENT

#EXECUTION TERMINATED
#

B29957